Lecture Notes in Artificial Intelligence 3464

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Sven A. Brueckner
Giovanna Di Marzo Serugendo
Anthony Karageorgos
Radhika Nagpal (Eds.)

Engineering Self-Organising Systems

Methodologies and Applications

1 3

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Sven A. Brueckner
Altarum Institute
3520 Green Court, Suite 300, Ann Arbor, MI 48105-1579, USA
E-mail: sven.brueckner@altarum.org

Giovanna Di Marzo Serugendo
University of Geneva, Centre Universitaire d'Informatique
24 rue Général-Dufour, 1211 Geneva 4, Switzerland
E-mail: Giovanna.Dimarzo@cui.unige.ch

Anthony Karageorgos
University of Thessaly, Department of Computer and Communication Engineering
37 Glavani - 28th October Str., Deligiorgi Building, 4th floor, room D3/4
382 21 Volos, Greece
E-mail: karageorgos@computer.org

Radhika Nagpal
Harvard University, Division of Engineering and Applied Sciences
Computer Science Dept., 235 Maxwell Dworkin, 33 Oxford Street, Cambridge
MA 02138, USA
E-mail: rad@eecs.harvard.edu

Library of Congress Control Number: 2005926500

CR Subject Classification (1998): D.2.11, C.2.4, C.2, D.2.12, D.1.3, D.4.3-4, H.3, H.4, K.4.4

ISSN	0302-9743
ISBN-10	3-540-26180-X Springer Berlin Heidelberg New York
ISBN-13	978-3-540-26180-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11494676 06/3142 5 4 3 2 1 0

Preface

The spread of the Internet, mobile communications and the proliferation of new market models, such as e-commerce, has resulted in the whole information infrastructure operating as a global dynamic system. The complexity and the inherent dynamism of the resulting global system require software capable of autonomously changing its structure and functionality to meet dynamic changes in the requirements and the environment without immediate human intervention. In particular, contemporary software applications must provide highly customised services to a huge user population by dynamically adapting to personal requirements. Furthermore, new maintenance approaches need to be followed, for example continuously running software should evolve on run-time to meet ever-changing user requirements. Finally, new ways for handling exceptions and component failure and replacement, as well as changes in the environment are required, for example as is the case in networks including large numbers of smart computing entities, such as ad hoc sensors and MEMs devices. In large interconnected software systems such tasks cannot be achieved by approaches involving direct supervision and centralised management.

A way to meet requirements of this kind is to utilise the emergent properties of distributed interacting software referring to concepts such as self-organisation, self-regulation, self-repair and self-maintenance. However, in artificial systems, environmental pressures and local interactions and control may lead to unpredicted or undesirable behaviour. Understanding how to engineer the correct self-organising behaviour is thus an issue of major concern.

Self-organising applications (SOAs) are able to dynamically change their functionality and structure without direct user intervention to meet changes in requirements and their environment. The overall functionality delivered by SOAs typically changes progressively, mainly in a nonlinear fashion, until it reaches (emerges to) a state where it satisfies the system requirements at the time, and therefore it is termed *self-organising* or *emergent* behaviour. Self-organising behaviour is often the result of the execution of a number of individual application components that locally interact with each other aiming to achieve their local goals, for example systems that are based on agents or distributed objects. The main characteristic of such systems is their ability to achieve complex collective tasks with relatively simple individual behaviours, without central or hierarchical control.

A major open issue is therefore how to engineer desirable self-organising behaviour in SOAs and how to avoid undesirable ones, given the requirements and the application environment. To address this issue, approaches originating from diverse areas such as nonlinear optimisation, knowledge-based programming and constraint problem solving are currently being explored. Furthermore, SOA engineers often take inspiration from the real world, for example from biology, chemistry, sociology and the physical world. Typical examples of SOAs are systems that reproduce socially based insect behaviour, such as ant-based systems, artificial life, or robots. Although the results achieved so far are promising, further work is required until the problem is sufficiently addressed.

This book is complementary to a sister volume published in 2003, which aimed at establishing the field of *Engineering Self-organising Systems* and it focused on the foundations of self-organising systems. This year the emphasis is on methodological aspects and on applications of self-organising approaches. The book comprises revised versions of papers presented at the Engineering Self-organising Applications (ESOA 2004) workshop, held during the Autonomous Agents and Multi-agent Systems conference (AAMAS 2004) in New York in July 2004, and selected invited papers from leading contributors in the self-organisation field.

Part I contains three papers related to state of the art of self-organising systems. Wolf and Holvoet review historical definitions of the terms self-organisation and emergence and provide new aggregated definitions of each term supported by examples. Subsequently, Bar Yam demonstrates the limitations of decomposition-based engineering for the development of highly complex systems using multi-scale analysis. Ulieru then discusses the characteristics of adaptive information infrastructures and their role in human/machine and hardware/software integration.

In Part II approaches to designing self-organising systems are presented. d'Inverno and Saunders provide a mathematical formalisation and discuss the advantages of using an agent-based approach to develop biologically plausible models of stem cell systems in the context of a case study. Subsequently, Bour et al. address the issue of the creation of visual ambiences based on the coordinated activity of tiny computing entities distributed randomly on a 2D canvas that can only change their own color and perceive their immediate neighbors. Edmonds argues on the use of adaptive approaches producing reliable self-organised software systems. The argument is supported by defining a class of simple multi-agent systems and showing that it can be evolved to perform simple tasks. Nowostawski et al. then propose an evolutionary computation model based on the theory of hypercycles and autopoiesis. Subsequently, Hales discusses the use of tag dynamics to realize adaptive node behaviour in P2P systems (selfish vs. altruistic) based on results of P2P simulations.

Part III describes applications of self-organisation in self-assembly and robotic systems. Mamei et al. present self-organising spatial shapes in mobile particles with minimal capabilities. Poulton et al. discuss a method for directed self-assembly of 2-dimensional mesoblocks using top-down/bottom-up design. Subsequently, Galstyan et al. present a stochastic model for adaptive task allocation in robots. Finally, White and Helferty discuss the application of division-of-labor principles to achieve emergent team formation in robot soccer.

In Part IV self-organisation models based on the use of stigmergy are discussed. Parunak and Brueckner discuss stigmergic learning for self-organising mobile ad hoc networks (MANETs). Karuna et al. propose a stigmergy-based approach for emergent forecasting in manufacturing coordination and control. Subsequently, Foukia takes inspiration from natural systems and proposes a self-organising approach for intrusion detction and response in networks. Along a similar line, Armetta et al. describe a self-organising model for managing dynamic flow in production chains.

Part V concludes the book with industrial applications of self-organising systems. Lauterbach et al. describe self-organisation and fault-tolerance issues in a wired peer-to-peer sensor network for textile applications. Subsequently, Brueckner and Gerth discuss the application of distributed adaptive optimisation techniques to digital car-body development. Finally, Graupner et al. propose adaptive service placement algorithms for autonomous service networks.

We are grateful to the Programme Committee of the ESOA 2004 workshop for their timely reviews, and their useful suggestions on improving the workshop. All papers submitted to the workshop were reviewed by three members of the Programme Committee.

December 2004 Sven Brueckner, Giovanna Di Marzo Serugendo
Anthony Karageorgos, Radhika Nagpal

Programme Committee

Table of Contents

Emergence Versus Self-organisation: Different Concepts but Promising When Combined

Tom De Wolf and Tom Holvoet

Department of Computer Science, Kuleuven,
Celestijnenlaan 200A, 3001 Leuven, Belgium
{Tom.DeWolf, Tom.Holvoet}@cs.kuleuven.ac.be

Abstract. A clear terminology is essential in every research discipline. In the context of ESOA, a lot of confusion exists about the meaning of the terms emergence and self-organisation. One of the sources of the confusion comes from the fact that a combination of both phenomena often occurs in dynamical systems. In this paper a historic overview of the use of each concept as well as a working definition, that is compatible with the historic and current meaning of the concepts, is given. Each definition is explained by supporting it with important characteristics found in the literature. We show that emergence and self-organisation each emphasise different properties of a system. Both phenomena can exist in isolation. The paper also outlines some examples of such systems and considers the combination of emergence and self-organisation as a promising approach in complex multi-agent systems.

1 Introduction

In the context of engineering self-organising applications there are two very important concepts to consider: emergence and self-organisation. In many multi-agent systems and complex adaptive systems in general, a combination of the two concepts is often used. As a consequence, much literature describes emergence and self-organisation incorrectly as synonyms and this results in misconception about their meaning. When engineering such applications, using a clear terminology is very important. To clarify the distinction between emergence and self-organisation, this paper's goal is to propose a working definition of both concepts. This definition is supported by characteristics that most literature describes as essential for emergence or self-organisation.

Emergence and self-organisation each emphasise very different characteristics of a system's behaviour. Both phenomena can exist in isolation and they can co-exist in a dynamical system. The first two sections of this paper describe each phenomenon separately by giving a historic overview of the use of each concept, proposing a working definition, and outlining their important characteristics to explain and support the definition given. The third section relates emergence and self-organisation to each other by discussing their similarities and differences.

This is illustrated with examples where each phenomenon occurs separately. After that, a section is devoted to the combination of both phenomena in a single system. Finally we conclude this paper.

2 Emergence

Typically, people describe 'emergence' as the phenomenon where global behaviour arises from the interactions between de local parts of the system. In most literature there is nothing more than this vague description. Examples of emergence around us are: global pheromone paths that arise from local path-following and pheromone-dropping ants, the swarming movement of a flock of birds, a traffic jam from the interactions of cars, etc.

The goal of this section is to develop a more detailed working definition for 'emergence'. First, a historic overview of the early use of the concept is given. The second part proposes a definition of emergence that is consistent with the given history and outlines the important characteristics found in literature.

2.1 Historic Overview

Emergence is not a new topic [1][1]. Conceptual constructs such as 'whole before its parts' (i.e. to consider an explanation in terms of the global behaviour more important than explaining how the system works in terms of local behaviour) and 'Gestalt' (i.e. a configuration or pattern of elements so unified as a whole that it cannot be described merely as a sum of its parts), which resemble emergence, can be found in western thought since the time of ancient Greeks.

However, 'whole before its parts' and 'Gestalt' refer to a pre-given coherent entity, whereas emergence is not pre-given but a dynamical construct arising over time. In the context of a dynamical system, the meaning of emergence is not new either. It was used over 100 years ago by the English philosopher G.H. Lewes in 1875. Lewes distinguished between 'resultant' and 'emergent' chemical compounds coming about from a chemical reaction [2]:

> (...) although each effect is the *resultant* of its components, we cannot always trace the steps of the process, so as to see in the product the mode of operation of each factor. In the latter case, I propose to call the effect an *emergent*. It arises out of the combined agencies, but in a form which does not display the agents in action (...). (italics added)

Lewes' term was borrowed during the 1920s to form the backbone of a loosely joined movement in the sciences, philosophy and theology known as emergent evolutionism or proto-emergentism [1]. The concept of emergence was hotly debated and mainly used against reductionism, which stated that a system can be reduced to the sum of its parts. Proto-emergentism had few answers when it came to understanding how emergence itself was possible, i.e. how the lower-level inputs are transformed to the higher-level outputs during emergence.

[1] The historic overview of emergence is based on [1].

A second movement, called neo-emergence or complexity theory [1], tries to address the lack of understanding emergence. The concept of emergence in complex systems has very diverse scientific and mathematical roots: cybernetics, solid state/ condensed matter physics, evolutionary biology, artificial intelligence, artificial life, etc. There are actually four central schools of research that each influences the way emergence in complex systems is studied:

– **Complex adaptive systems theory**, which became famous at the Santa Fe Institute and which explicitly uses the term 'emergence' to refer to the macro-level patterns arising from interacting agents (see [3], [4], and [5]);
– **Nonlinear dynamical systems theory and Chaos theory**, which promulgates the central concept of attractors, i.e. a specific behaviour to which the system evolves. One kind of attractor is the so called strange attractor that the philosopher of science David Newman (1996)[6] classifies as an authentically emergent phenomenon.
– **The synergetics school**, which initiated, among others, the study of emergence in physical systems. They describe the idea of an order parameter that influences which macro-level coherent phenomena a system exhibits [7].
– **Far-from-equilibrium thermodynamics**, which was introduced by Ilya Prigogine and which refers to emergent phenomena as dissipative structures arising at far-from-equilibrium conditions[8].

In short, the uses of the concept of emergence refer to two important characteristics: a global behaviour that arises from the interactions of the local parts, and that global behaviour cannot be traced back to the individual parts.

2.2 A Working Definition

It is important that the concept of emergence is used consistently in literature. In the first place we need to be consistent with the historic use of the concept, as outlined above. In current literature, this is not such a big problem w.r.t. emergence. There is a larger misconception about the meaning of self-organisation, which is discussed later. The definition that we propose as a working definition for emergence is:

> A system exhibits emergence when there are coherent emergents at the macro-level that dynamically arise from the interactions between the parts at the micro-level. Such emergents are novel w.r.t. the individual parts of the system.

The definition above uses the concept of an 'emergent' as a general term to denote the result of the process of emergence: properties, behaviour, structure, patterns, etc. The 'level' mentioned refers to certain points of view. The macro-level considers the system as a whole and the micro-level considers the system from the point of view of the individual entities that make up the system.

This definition resulted from an extensive literature study, which identified the most important characteristics found in literature. The remainder of this part outlines these characteristics in order to explain the different aspects of the proposed definition in more detail.

Micro-Macro effect [3, 9, 10, 1, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21]. This is the most important characteristic and is mentioned explicitly in most literature. A micro-macro effect refers to properties, behaviours, structures, or patterns that are situated at a higher macro-level and arise from the (inter)actions at the lower micro-level of the system. We call such properties 'emergents'. In other words, the global behaviour of the system (i.e. the emergent) is a result from the interactions between the individual entities of the system.

Radical Novelty [9, 11, 1, 22, 17, 19, 10, 20, 21, 13]. The global behaviour is novel w.r.t. the individual behaviours at the micro-level, i.e. the individuals at the micro-level have no explicit representation of the global behaviour. In terms of reductionism this is formulated as: the macro-level emergents are not reducible to the micro-level parts of the system (= non-reductionism). In literature there are various formulations: 'not directly described by' [9, 10], 'can not be reduced to' [11], 'neither predictable nor deducible from' [1], 'without reference to the global pattern' [17], 'the whole is greater than the sum of its parts' [13].

From [22] we learn that we must pay attention. Stating that emergents are not captured by the behaviour of the parts is a serious misunderstanding. Radical novelty arises because the collective behaviour is *not readily understood from* the behaviour of the parts. The collective behaviour is, however, implicitly contained in the behaviour of the parts if they are studied in the context in which they are found. Emergent properties cannot be studied by physically taking a system apart and looking at the parts (=reductionism). They can, however, be studied by looking at each of the parts in the context of the system as a whole.

Coherence [1, 14, 13, 12, 22, 16]. Coherence refers to a logical and consistent correlation of parts. Emergents appear as integrated wholes that tend to maintain some sense of *identity* over time (i.e. a *persistent* pattern). Coherence spans and correlates the separate lower level components into a higher level unity, i.e. correlations between components are needed to reach a coherent whole [22]. This coherence is also called 'organisational closure' [12].

Interacting Parts [13, 17, 18, 14, 12]. The parts need to interact - parallelism is not enough. Without interactions, interesting macro-level behaviours will never arise. The emergents arise from the interactions between the parts.

Dynamical [1, 12, 3, 13, 17, 10, 20]. In systems with emergence, emergents arise as the system evolves in time. Such an emergent is a new kind of behaviour that becomes possible at a certain point in time. Therefore, as a dynamical construct we can relate the appearance of emergents to the appearance of new attractors in dynamical systems, i.e. bifurcations [1, 12].

Decentralised Control [13, 12, 16]. Decentralised control is using only local mechanisms to influence the global behaviour. There is no central control, i.e. no single part of the system directs the macro-level behaviour. The actions of the parts are controllable. The whole is not directly controllable. This characteristic is a direct consequence of the radical novelty that is required for emergence. Centralised control is only possible if that central part of the system has a representation of the global behaviour (e.g. a plan).

Two-Way Link [13, 20, 21]. In emergent systems there is a bidirectional link between the macro-level and the micro-level. From the micro-level to the macro-level, the parts give rise to an emergent structure (see ‘micro-macro effect’ above). In the other direction, the emergent structure influences its parts. Higher level properties have causal effects on the lower level, i.e. downward causation. For example, path-formation with ants: the emergent path influences the movement of the micro-level ants because they follow the pheromones.

Robustness and Flexibility [13, 12]. The need for decentralised control and the fact that no single entity can have a representation of the global emergent, implies that such a single entity cannot be a single point of failure. Emergents are relatively insensitive to perturbations or errors. Increasing damage will decrease performance, but degradation will be ’graceful’: the quality of the output will decrease gradually, without sudden loss of function. The failure or replacement of a single entity will not cause a complete failure of the emergent. This flexibility makes that the individual entities can be replaced, yet the emergent structure can remain. For example, birds in a flock or cars in a traffic jam can be replaced by other birds or cars, yet the flock and traffic jam phenomena remain.

3 Self-organisation

An intuitive and linguistic definition of self-organisation given by Dempster in 1998 [23] is: “Self-organisation refers to exactly what is suggested: systems that appear to organise themselves without external direction, manipulation, or control.” The ‘organisation’ is related to an increase in the structure or order of the system behaviour. Like the section about emergence, this section develops a more detailed working definition. An example of self-organisation is: ad-hoc networks that autonomously built their structure as network devices detect each other’s presence.

3.1 Historic Overview

> *[Consider] what would happen in a new world, if God were now to create somewhere in the imaginary spaces matter sufficient to compose one, and were to agitate variously and confusedly the different parts of this matter, so that there resulted a chaos as disordered as the poets ever feigned, and after that* ***did nothing more than lend his ordinary concurrence to nature, and allow her to act in accordance with the laws which He had established*** *... . I showed how the greatest part of matter of this chaos must, in accordance with these laws, dispose and* ***arrange itself*** *in such a way as to present the appearance of heavens; how in the meantime some of its parts must compose an earth and some planets and comets, and others a sun and fixed stars.*(René Descartes, 1637 [24], part 5)

The notion of spontaneous, dynamically-produced organisation is very old [2]. This is illustrated in the quotation above from [24], which captures the essence of self-organisation. The phenomenon is only called "self-organisation" in the years after the Second World War, in communities connected with cybernetics and computing machinery [26, 27]. The first appearance of the term seems to be in a 1947 paper by W. Ross Ashby [28].

Remarkably, Ashby gave a pretty clear explanation of what he meant by 'organisation': the organisation of a system is the functional dependence of its future state on its present state and its external inputs, if any. Ashby understood a system to be self-organising if the system changed its own organisation, rather than being changed by an external entity. Ashby's description closely matches what we will define as self-organisation later.

The main research domains, where self-organisation was studied after its introduction, were physics, computer science, and systems theory. In the physical sciences self-organisation was extensively applied, from the 1970s onwards, to pattern formation[29] and spontaneous symmetry breaking [30] and to cooperative phenomena [31]. There has been confusion about what self-organisation actually is. For example, [32] claimed that the transition from lamellar to turbulent flow is an instance of self-organisation. Others have just as vigorously denied this. There has been no resolution of the controversy, and no means to resolve it [33]. In any case, just like with emergence there is confusion about the meaning of "self-organisation".

Within computer science, the primary applications have been to learning [34, 26]; to adaptation [35]; and to "emergent" or distributed computation [36, 37]. Also in economics [38, 39], and in ecology [40, 17], self-organisation has begun to feature, complete with the now-expected disputes about whether certain processes are self-organising.

In the 1980s, self-organisation became one of the ideas, models and techniques bundled together as the "sciences of complexity" [41]. This bundle has been successful at getting itself adopted by some researchers in essentially every science, so the idea of self-organisation is now used in a huge range of disciplines.

One of those disciplines is multi-agent systems. Multi-agent systems are used to model self-organising systems. Cooperation [42] and group formation [43] in multi-agent systems make the system more organised, which is done autonomously by the agents. A number of self-organising applications are realised [44], such as in networks [45], in robotics, and the self-organisation of a vocabulary between agents [46].

3.2 A Working Definition

It is important that the concept of self-organisation is used consistently in literature. In the first place we need to be consistent with the historic use of the concept, as outlined above. Therefore 'autonomy' and 'increase in structure' should be included. In current literature, there is a often misconception about

[2] The historic overview of self-organisation is based on the Ph.D. of C.R.Shalizi [25].

the meaning of self-organisation. For example, in [17, 16] the authors define self-organisation when they actually define emergence according to our definition. The definition that we propose as a working definition for self-organisation is:

> Self-organisation is a dynamical and adaptive process where systems acquire and maintain structure themselves, without external control.

The 'structure' can be a spatial, temporal or functional structure. 'No external control' refers to the absence of direction, manipulation, interference, pressures or involvement from outside the system. This does not exclude data inputs from outside the system as long as these inputs are not control instructions. Note, the identification of the 'boundary' of the system is extremely important when deciding if a system is self-organising or not. It is important to specify what we consider as an external control and what not.

An extensive literature study identified the characteristics, considered important in literature. Below we outline these characteristics in order to explain the different aspects of the proposed definition in more detail.

Increase in Order [25, 47, 20, 12, 11, 45, 15, 19, 44]. One important characteristic of self-organisation is the 'organisation' part of the concept. [20] describes organisation as the arrangement of selected parts so as to promote a specific function. This restricts the behaviour of the system in such a way as to confine it to a smaller volume of its state space. This smaller region of state space is called an attractor. In essence, organisation can be looked at as an increase in the order of the system behaviour which enables the system to acquire a spatial, temporal, or functional structure. Note that not every system that has an increase in order needs to be self-organising. Complete autonomy of the behaviour is also needed (see below).

In [25], a more formal approach is used to define self-organisation. The author uses the notion of statistical complexity to denote the order mentioned in this paper. An increase in statistical complexity is considered a necessary condition for self-organisation. Statistical complexity measures the average amount of historical memory stored in the process. This formulation covers a number of other definitions found in literature. For example, 'the arrangement of selected parts' implies that the arrangement is a kind of historic memory of the process that becomes bigger when more and more parts are arranged.

An increase in order implies that such systems start from semi-organised or completely random initial conditions [44] (i.e. no historical memory). What is also possible is that a system behaviour becomes less ordered (i.e. looses historical memory) as a result from a change. Both situations leave room for an increase in order through the process of self-organisation.

The formulation 'as to promote a specific function' in [20] is important. A system with no order can not exhibit useful behaviour. But also a system with too much order can have this problem. It is possible that processes organise themselves into conditions so complex that no usable functionality can result from it. In other words, there can be too much historical memory. The systems in between, i.e. at the edge of order and chaos [48, 49], can exhibit a more flexible and organised behaviour. Therefore, self-organisation needs to find a balance between no order and too much order.

Autonomy [25, 47, 20, 12, 11, 45, 17, 19, 44]. Not every increase in order is self-organising. The second important characteristic of self-organisation is the absence of external control ('self'). A system needs to organise without interference from the outside. Other formulations are: 'without an external agent imposing it', 'spontaneous, i.e. not steered by an external system', 'the constraints on form (i.e. organisation) of interest to us are internal to the system', etc.

Does the lack of external control and autonomy mean that such a system can have no input at all? Of course not, in general, input is still possible as long as the inputs are no control instructions from outside the system. In other words, normal data input flows are allowed but the decision on what to do next should be made completely inside the system, i.e. the system is autonomous. For example, plugging in a PnP device in a computer can be considered as normal data input. A self-organising behaviour could be the autonomous configuration of drivers by the computer system. If a user has to install the drivers himself then there is no self-organisation.

The notion of 'boundary of a system' becomes very important here. To be able to say if a certain system is self-organising, we must first clearly define the boundary of the system. We need to separate the inside from the outside.

Adaptability or Robustness w.r.t. changes [1, 45, 12, 44]: In self-organising systems, robustness is used in terms of adaptability in the presence of perturbations and change. A self-organising system is expected to cope with that change and to maintain its organisation autonomously. In other words, a self-generated, adaptable behaviour is needed [1], and taking into account past experiences can be helpful [45]. [44] formulates this adaptability as: "a change in the environment may influence the same system to generate a different task, without any change in the behavioural characteristics of its constituents".

This adaptability implies the need for the system to be able to exhibit a large variety of behaviours. Self-organisation requires the evolution towards a certain attractor in state space (i.e. towards a certain organised behaviour). There are different kinds of attractors, from a point attractor that allows only one behaviour, a limit cycle that allows periodic behaviour, towards a chaotic attractor that allows a very large variety of behaviours. To be adaptable, the system needs to make a selection between behaviours and at the same time consider a variety of behaviours [12]. Too much variety, like the chaotic attractor, makes the system uncontrollable. Too much selection, like the point attractor, results in a system that is not flexible enough. This is related to balancing the system on the edge of order and chaos [48, 49] in order to be able to promote a specific function (see 'Increase in order'). For example, a system's initial conditions may support many functions (i.e. chaotic attractor), but there need to be selective pressures to focus the outcome [19]. For example, a system that has a chaotic attractor can balance its behaviour on a specific part of that attractor.

Dynamical, i.e. far-from-equilibrium [25, 1, 11, 12, 50]: An essential property of self-organisation is that it is a process. Over time, there is an increase in order, i.e. a dynamic towards more order.

Related to the required adaptability in a rapidly changing context, self-organising behaviour needs to be dynamic. Changes influence the organised structure. In order to maintain that structure, there needs to be a constant dynamic that handles these changes. In other words, the system needs to be far-from-equilibrium in order to maintain the structure. Prigogine [50] considers far-from-equilibrium as one of the mathematically deduced requirements. A far-from-equilibrium system is more fragile and sensitive to changes in the environment, but also more dynamic and capable to react.

4 Comparing Emergence and Self-organisation

To summarise, the essence of emergence is the existence of a global behaviour that is novel w.r.t. the constituent parts of the system. The essence of self-organisation is an adaptable behaviour that autonomously acquires and maintains an increased order (i.e. statistical complexity, structure, ...). In this section we describe the similarities and the differences between both concepts.

4.1 Similarities

Because emergence and self-organisation each emphasise very different aspects of the system behaviour there are few similarities. The main similarity is that emergence and self-organisation are both dynamic processes arising over time. Both are also robust. However, emergence is robust w.r.t. the flexibility in the specific parts that cause the emergent properties (i.e. the failure of one single part will not result in a complete failure of the emergent property). Self-organisation is robust w.r.t. the adaptability to change and its ability to maintain the increased order. Having few similarities does not exclude that both concepts are related to each other. They complement each other when combined (see below).

4.2 Differences

The sections above show that emergence and self-organisation each emphasise different characteristics of a system. Both concepts can exist in isolation, which is discussed here. First we consider self-organisation without emergence, and then emergence without self-organisation is described and illustrated with examples.

Self-organisation without Emergence. Figure 1(a) schematically illustrates a system with self-organisation, but no micro-macro effect. There are no controls that come from outside the boundary of the system. The curved arrow

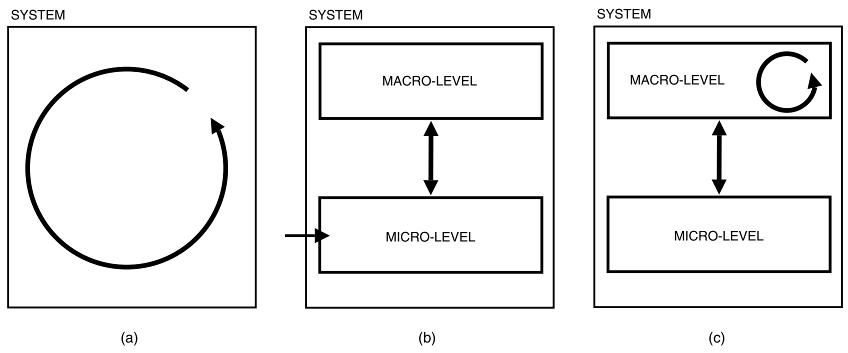


Fig. 1. (a) self-organisation without emergence; (b) emergence without self-organisation; (c) Combining Emergence and Self-Organisation

represents the internal organising process. The properties that are specific for emergence, but not needed for self-organisation, are radical novelty, micro-macro effect, flexibility w.r.t. the entities, and decentralised control. When one of these properties is not present we have no emergence.

Consider certain kinds of multi-agent systems, called a 'classical' multi-agent system in [19]. Such a system is autonomous and increases its order through interactions. However, there is no need for the system to exhibit emergent properties, i.e. properties that are novel w.r.t. the agents in the system. When, for example, every agent has a model of the global behaviour that has to be achieved, this behaviour is explicitly present in the parts of the system and thus not novel.

A system where there is a single controlling agent that directs the global behaviour (i.e. no decentralised control), needs an explicit plan in that controlling agent. Of course, a self-organising process can re-elect a controlling agent when other agents become more appropriate for the job, but there is no radical novelty.

Another important property of emergent systems is 'graceful degradation' because of the flexibility w.r.t. the entities. A single entity is not essential for the functioning of the whole system. A self-organising system where each entity is essential does not conform with the needed characteristics of emergence.

Emergence without Self-organisation. Figure 1(b) schematically illustrates the other situation. The system has a micro-macro effect, but it is not self-organising. The essential properties here are the increase in order, no external control and adaptability.

Emergence without self-organisation is definitely possible. For example in physics, thermodynamics can emerge from statistical mechanics in a stationary (and so non-self-organising) system [25]. A stationary process is a process where the order is time-translation invariant, i.e. no increase in order. Consider a gas material that has a certain volume in space. This volume is an emergent property that results form the interactions (i.e. attraction and repulsion) between the individual particles. However, such a gas is in a stationary state. The statistical complexity remains the same over time, i.e. the particles can change place but the amount of structure remains the same. In this case, we have a system whose initial conditions are enough to exhibit emergent properties.

Adaptability refers to the need to reach a balance between selection of a specific behaviour and the consideration of a large variety of behaviours. [45] formulates this in terms of a balance between exploration and exploitation. A system can exhibits chaos (i.e. considering a large variety of behaviours and also constantly switching between these) that emerged from the interactions between the micro-level parts. But, such a system is not self-organising because it does not organise itself to promote a specific function.

5 Combining Emergence and Self-organisation

In most systems that are considered in literature, emergence and self-organisation occur together. Research in the multi-agent community and the complex adaptive systems community focuses on such systems. In very complex (multi-agent) systems, i.e. distributed, open, large, situated in a dynamic context, etc., the combination of emergence and self-organisation is recommended. In a complex (multi-agent) system there is often a need to keep the individual entities relatively simple (e.g. for scalability). Self-organisation requires an increase in order that promotes a certain function or property. Simple individuals cannot direct such a complex system, so the global coherent behaviour should emerge from the interactions between the individuals. The other way around, a complex (multi-agent) systems can be required to exhibit emergent behaviour. Because of the complexity, it is impossible to impose an initial structure on such a system that results in an emergent property. The only possibility to get a coherent behaviour at the macro-level is to let that behaviour arise and organise autonomously, i.e. self-organisation. Thus, combining both phenomena is a promising approach to engineer a coherent behaviour for complex (multi-agent) systems.

Combining self-organisation and emergence in one system imposes the question on how both phenomena should be linked to each other. To answer this, there are multiple point-of-views possible. A first point of view considers self-organisation as a cause, i.e. emergent properties in complex systems are the result of a self-organising process[11, 17, 51], possibly combined with selective pressures towards a certain emergent behaviour[49]. Thus, the interactions between the individual entities are the self-organisation. Self-organisation is situated at the micro-level of the emergent process. A second point of view considers self-organisation as an effect, i.e. emergence results in self-organisation. Thus, self-organisation is an emergent property. Figure 1(c) schematically illustrates what is stated in [15]: "... self-organising behaviour occurs at the macro-level". This point-of-view is explained as a result of a characteristic of self-organisation, i.e. the need for an increase in order. In an emergent system, at the micro-level the dynamics are often very complicated and disordered. This means an increase in order can only occur within the global level [19], i.e. the emergents become more and more organised. The author of [19] also states that the system as a whole is decreasing its order. The reason for this is that at the micro-level of emergent systems the dynamics are often very complicated and disordered. Also in [25] the author states: "... self-organisation increases [statistical] complexity, while emergence, generally speaking, reduces it ...". Thus, this need for increased order seems to conform with self-organisation as an effect at the macro-level of emergence.

Because emergence and self-organisation are often described in combination with each other, a characteristic, that some authors ascribe to one of both phenomena, is probably more specific for the combination of the phenomena. This characteristic is *Nonlinearity* [1, 12, 50, 17]: A system, without a priori order and where the emergence has to be self-organised, requires the "small cause, large effect" principle and should have an intense focus on nonlinear interactivity. Nonlinearity enables those secondary effects at the macro-level that we call emergents. This nonlinearity is often achieved through positive feedback that amplifies an initial change. The result of the first amplification again triggers positive feedback that amplifies the effect of the change. After a while, a number of components have 'aligned' themselves with the configuration created by the initial change and the configuration stops growing: the system has 'exhausted' the available resources. This alignment is often the emergent property of the system. This way, an emergent can self-organise.

Nonlinear mechanisms are related to one of the properties of self-organisation, mathematically deduced by Prigogine [50]. He states that at least one of the components in the system must exhibit auto-catalysis. A system exhibits auto-catalysis if one of its components is causally influenced by another component, resulting in its own increase. Actually, auto-catalysis is a kind of positive feedback (e.g. pheromone reinforcements by ants) that can cause a nonlinear effect.

In a self-organising system, the emergence should be adaptive in order to have a system that self-organises in the presence of a changing situation. When there has been a nonlinear 'alignment' with positive feedback, the only possibility to escape that alignment, and end up in a new alignment that is adapted to the new situation, is to use negative feedback. In more complex self-organising systems, there will be several interlocking positive and negative feedback loops, so that changes in some directions are amplified while changes in other directions are suppressed. In [17] the presence of positive and negative feedback is also considered important for adaptive behaviour.

6 Conclusion

The starting point of this paper was that it is important to use a clear terminology when engineering self-organising applications. The discussion showed that the important concepts of emergence and self-organisation refer to two distinct phenomena. They each emphasise different characteristics of a system. Confusion in literature should be avoided by using each concept correctly and certainly not as synonyms. Emergence emphasises the presence of a novel coherent macro-level emergent (property, behaviour, structure, ...) as a result from the interactions between micro-level parts. Self-organisation emphasises the dynamical and adaptive increase in order or structure without external control.

Both phenomena can exist in isolation, yet a combination of both phenomena is often present in complex dynamical systems. In such systems, the complexity is huge, which makes it infeasible to impose a structure a priori: the system needs to self-organise. Also, the huge number of individual entities imposes a need for emergence. For scalability we can not put an entire plan for the global structure in a single entity; we need to keep the individuals rather simple and let the complex behaviour self-organise as an emergent behaviour from the interactions between these simple entities. A combination of emergence and self-organisation, which is already applied in literature [19, 44, 52], is a promising approach to engineer large-scale multi-agent systems.

This paper presents results from research sponsored by the research council of the K.U.Leuven. The results have been obtained in the Concerted Research Action on Agents for Coordination and Control - AgCo2 project.

References

1. Goldstein, J.: Emergence as a construct: History and issues. Emergence **1** (1999)
2. Lewes, G.: Problems of Life and Mind. Volume 2. Kegan Paul, Trench, Turbner, London (1875)
3. Holland, J.: Emergence: from Chaos to Order. Addison-Wesley (1998)
4. Kauffman, S.: At Home in the Universe: the Search for the Laws of Self-Organization and Complexity. Oxford University Press (1995)
5. Langton, C.: Studying artificial life with cellular automata. In Farmer, D., Lapedes, A., Packard, N., Wendroff, B., eds.: Evolution, Games, and Learning: Models for Adaptation in Machines and Nature, Proceedings of the Fifth Annual Conference of the Center for Nonlinear Studies. (1986)
6. Newman, D.: Emergence and strange attractors. Philisophy of Science **36** (1996)
7. Haken, H.: The Science of Structure: Synergetics. Van Nostrand Reinhold, NY (1981)
8. Nicolis, G.: Physics of far-from-equilibrium systems and self-organization. In Davies, P., ed.: The New Physics. Cambridge University Press (1989)
9. Crutchfield, J.: Is anything ever new? considering emergence. Working Paper 94-03-011, Santa Fe Institute (1994)
10. Crutchfield, J.: The calculi of emergence: Computation, dynamics, and induction. Working Paper 94-03-016, Santa Fe Institute (1993)
11. Heyligen, F.: Self-organization, emergence and the architecture of complexity. In: Proceedings of the 1st European Conference on System Science, Paris (1989)
12. Heyligen, F.: The science of self-organisation and adaptivity. In: The Encyclopedia of Life Support Systems. UNESCO Publishing-Eolss Publishers (2002)
13. Odell, J.: Agents and complex systems. JOT **1** (2002) 35–45
14. Odell, J.: Objects and agents compared. JOT **1** (2002) 41–53
15. Parunak, H.V.D., Brueckner, S.: Entropy and self-organization in multi-agent systems. In: Proceedings of the Fifth International Conference on Autonomous Agents, ACM Press (2001) 124–130

16. Engineering self-organizing applications workgroup - mission statement (2003) (preliminary version available at http://cui.unige.ch/˜dimarzo/esoawg/mission.pdf).
17. Camazine, S.: Self-Organization in biological systems. Princeton Studies in Complexity. Princeton Univ Press (2001)
18. Parunak, H., Brueckner, S., Sauter, J.: ERIM's Approach to Fine-Grained Agents. In: In Proceedings of NASA/JPL Workshop on Radical Agent Concepts (WRAC'02). (2002) (available at http://www.erim.org/˜vparunak/.
19. Parunak, H.D., Brueckner, S.A.: Engineering Swarming Systems. In Bergenti, F., Gleizes, M.P., Zambonelli, F., eds.: Methodologies and Software Engineering for Agent Systems. Kluwer (2004) (to appear, available online at http://www.erim.org/˜vparunak/).
20. Self-organizing systems faq. online (2003) available at $http://www.calresco.org/sos/sosfaq.htm.$
21. Lucas, C.: Emergence and evolution - constraints on form. online (2003) (available at $http://www.calresco.org/emerge.htm$).
22. Bar-Yam, Y.: 0, Overview: The Dynamics of Complex Systems - examples, questions, methods and concepts. Studies in Nonlinearity. In: Dynamics of Complex Systems. Westview Press (1997)
23. Dempster, M.B.L.: A Self-Organising Systems Perspective on Planning for Sustainability. Master's thesis, University of Waterloo, School of Urban and Regional Planning (1998) (online at $http://www.fes.uwaterloo.ca/u/mbldemps/pubs/$).
24. Descartes, R.: Discours de la méthode pour bien conduire sa raison, et chercher la vérité dans les sciences. In: The Philosophical Writings of Descartes, 1985, Cambridge University Press. Volume I., Leiden (1637) 111–151 translated in Discours on the Method of rightly conducting one's reason and seeking truth in the sciences.
25. Shalizi, C.R.: Causal Architecture, Complexity and Self-Organization in Time Series and Cellular Automata. PhD thesis, University of Wisconsin at Madison (2001) available at http://cscs.umich.edu/˜crshalizi/thesis/.
26. Yovits, M.C., Cameron, S., eds.: Self-Organising Systems: Proceedings of an Interdisciplinary Conference, Oxford, Pergamon Press (1960) vol. 2 of International Tracts in Computer Science and Technology and Their Application.
27. Von Foerester, H., Jr., G.Z.Z., eds.: Principles of Self-Organization: Transactions of the University of Illinois Symposium on Self-Organization, June 1959, New York, Information Systems Branch, U.S. Office of Naval Research, Pergamon Press (1962)
28. Ashby, W.R.: Principles of self-organizing dynamic systems. Journal of General Psychology **37** (1947) 125–128
29. Ball, P.: The Self-Made Tapestry: Pattern Formation in Nature. Oxford University Press (1999)
30. Nicolis, G., Prigogine, I.: Self-Organization in Nonequilibrium Systems: From Dissipative Structures to Order through Fluctuations. Wiley, New York (1977)
31. Haken, H.: Synergetics: An introduction: Nonequilibrium Phase Transitions and Self-Organization in Physics, Chemistry, and Biology. Springer Verlag (1977)
32. Klimontovich, Y.L.: Turbulent Motion and the Structure of Chaos: A New approach to the Statistical Theory of Open Systems. Kluwer Academic (1990/1991)
33. Frisch, U.: Turbulence: The Legacy of A.N. Kolmogorov. Cambridge University Press, Cambridge, England (1995)
34. Selfridge, O.G.: Pandemonium: A paradigm for learning. In Blake, D., Uttley, A., eds.: The Mechanisation of Thought Processes. Volume 10 of National Physical Laboratory Symposia. Her Majesty's Stationary Office, London (1959) 511–529

35. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. 2 edn. MIT Press, Cambridge, Massachusetts (1992) First edition in 1975.
36. Forrest, S., ed.: Emergent Computation: Self-Organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks: Proceedings of the Ninth Annual International Conference of the Center of Nonlinear Studies, Los Alamos, New Mexico, 1989, Amsterdam, North Holland (1990)
37. Crutchfield, J.P., Mitchell, M., eds.: The Evolution of Emergent Computation, Proceedings of the National Academy of Sciences. Volume 92. (1995)
38. Krugman, P.R.: The Self-Organizing Economy. Blackwell, Oxford (1996)
39. Shalizi, C.R.: Review of krugman, p.r. (1996) 'the self-organizing economy'. In: The Bactra Review. Volume 11. (1996) available at www.santafe.edu/˜shalizi/reviews/self-organizing-economy/.
40. Arthur, W.: The Green Machine: Ecology and the Balance of Nature. Basil Blackwell, Oxford (1990)
41. Pagels, H.R.: The Dreams of Reason: The Computer and the Rise of the Sciences of Complexity. Simon and Schuster, New York (1988)
42. Steels, L.: Cooperation between distributed agents through self-organisation. In: Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Elsevier Science Publishers Holland (1990) 175–196
43. Ghanea-Hercock, R.: Spontaneous group formation in multi-agent systems. In: Proceedings of Workshop on Self-Organisation in Multi-Agent Systems. (2000)
44. Mostefaoui, S.K., Rana, O.F., Foukia, N., Hassas, S., Marzo, G.D., Aart, C.V., Karageorgos, A.: Self-Organising Applications: A Survey. In: Proceedings of the International Workshop on Engineering Self-Organising Applications 2003. (2003) (extended version in 'Engineering Self-Organising Systems - Nature-Inspired Approaches to Software Engineering', ISBN: 3-540-21201-9, Springer Verlag).
45. Foukia, N., Hassas, S.: Towards self-organizing computer networks: A complex system perspective. In: Proceedings of the International Workshop on Engineering Self-Organizaing Applications 2003, Melbourne, Austrialia (2003)
46. Oudeyer, P.Y.: Self-organisation of a lexicon in a structured society of agents. In Floreano, D., Nicoud, J.D., Mondada, F., eds.: Advances in Artificial Life (ECAL 99). LNAI 1674, Berlin, Springer-Verlag (1999) 726–729
47. Haken, H.: Information and Self-Organisation: A Macroscopic Approach to Complex Systems. Springer Verlag, germany (1998)
48. Langton, C.G.: Computation at the edge of chaos: Phase transitions and emergent computation. Physica D **42** (1990) 12–37
49. Kauffman, S.: The Origins of Order: Self-Organization and Selection in Evolution. Oxford University Press (1993)
50. Glansdorff, P., Prigogine, I.: Thermodynamic study of structure, stability and fluctuations. Wiley, New York (1978)
51. Mamei, M., Zambonelli, F.: Self-organisation in multi-agent systems: a middleware approach. In: Proceedings of the International Workshop on Engineering Self-Organising Applications 2003. (2003)
52. Hadeli, Valckenaers, P., Zamfirescu, C.B., Brussel, H.V., Germain, B.S., Holvoet, T., Steegmans, E.: Self-organising in multi-agent coordination and control using stigmergy. In: Proceedings of the International Workshop on Engineering Self-Organising Applications 2003. (2003)

About Engineering Complex Systems: Multiscale Analysis and Evolutionary Engineering

Yaneer Bar-Yam

New England Complex Systems Institute,
Cambridge, MA 02138

Abstract. We describe an analytic approach, multiscale analysis, that can demonstrate the fundamental limitations of decomposition based engineering for the development of highly complex systems. The planning based process is limited by the interdependence of components and communication between design teams. Thus, the construction of many highly complex systems should be pursued by strategies modeled after biological evolution, or market economies, where extensive planning is forsaken and multiple parallel design efforts compete for adoption through testing in actual use.

1 Introduction

The recognition that highly complex system design and engineering requires new insights and tools has become a topic of increasing interest and importance as the number of active elements in systems and the real time demands on a system increase [1-5].

One of the central realizations about highly complex systems is that analysis and synthesis do not follow the same process. This is dramatically different from the case with conventional engineering analysis and design. When a system is sufficiently simple, analysis and synthesis occur by decomposition. Each part is understood, and the function of the entire system can be recognized through a composition process of the parts. When a system is highly complex this approach is not possible [1-3].

We have developed an analytic approach to the study of complex systems called Multiscale Analysis [1,6-13] that directly addresses the complexity of the system and its relationship to structure and function. This approach provides basic insight into design trade-offs. However, it also enables us to demonstrate quantitatively that design by decomposition strategies is unable to create systems beyond a certain level of complexity. This level is limited by the ability of a single agent (i.e. a human being) to understand the interdependencies between the components. When higher levels of complexity are necessary in order to design systems it is necessary to transition to an alternative synthesis strategy. This is the strategy of evolutionary engineering.

Evolutionary engineering abandons many of the highly valued conventional systems engineering strategies of well planned and fully understood system. It replaces these with the creation of a planned environment that fosters learning by doing and enables unanticipated advances. This approach is the natural strategy for developing highly complex systems because their behavior is ultimately untestable, discovery is a key part of ongoing improvement, and the necessary time scale for use and improvement is far shorter than what can be achieved by traditional cycles of planning and implementation. The false sense of security in planning is inferior to the recognition that the right environment is a better guarantee of rapid improvement and innovation.

Aspects of the evolutionary approach we describe [1-3] can be found in various more traditional and recent approaches. Incremental engineering [14] and experience based learning [15,16] are very traditional approaches in certain contexts. Recent extensions include spiral development and evolutionary acquisition [17] and adaptive programming [18]. A discussion of various engineering approaches in relation to a conventional understanding of evolution is provided in Ref. [19]. There are key differences between the evolutionary approach we describe and other strategies. These include an emphasis on parallel competitive development teams and the importance of creating an ongoing fielded implementation strategy where coexistence of multiple types of components are possible. This evolutionary process is most commonly associated with the formation of complex biological organisms. A free market system is also an example of an evolutionary system with particular features that are not present in all evolutionary contexts.

In this paper we will describe briefly key concepts from multiscale analysis. We will focus on their implications (1) for design decisions and (2) that limit the possibility of decomposition based design. Then we will describe historical experience with large engineering projects, and some of the steps we have taken toward defining an enlightened evolutionary engineering strategy.

2 Multiscale Analysis

Multiscale analysis [1,10] builds on the twin recognitions that scale and variety / complexity are both necessary for effective performance of systems:

- Scale: A task requires a system to have sufficient "scale" of action. Here scale refers to the number of elementary components that are coordinated in order to perform a task.
- Variety: A task requires a system to have sufficiently many distinct actions it can take. Variety is measured as the logarithm of the number of distinct actions that can be taken in a specified interval of time.

To explain these two issues in an intuitive way: it is possible to be effective at some tasks by brute force, and at others by carefully choosing the right action to take. When designing a system for its tasks, recognizing the degree to which scale and complexity play a role in the design of the system is also directly relevant to the process of design.

To understand the design implications of this analysis conceptually we note that when components are acting in a coordinated way, they cannot act independently. When high variety is required then components must be able to act independently. When scale is required then components must act coherently. Thus there are various degrees of tradeoff that are possible to achieve a particular amount of variety at each scale of action.

The key to multiscale analysis of the variety of any system is that each of the components has a limit on its variety—the logarithm of the number of distinguishable states. Components can be individuals that act in performing tasks, or individuals that manage or coordinate tasks or serve as communication channels. We do not assume anything *a priori* about the specific tasks or actions of the components. They can be the same or different from one another. The key is that each of them has a bound on its variety. If we have a system that is formed of many components, and some of these components are responsible for coordinating other components, then we can establish limits on what particular organizational structures can do. It may be that the variety associated with the coordination exceeds the variety of the components. This is true even if the components that must be coordinated are relatively simple. It is also true if the components have a high variety. The key is that quite generally, for a system of N components, the coordination may require of order N times the variety of the individual components, even in a fixed configuration of coordination. This means we may need N coordinating entities.

To understand the organizational limitations that are established by such an analysis consider a hierarchical system. We can consider as a hierarchical structure either a human organization with hierarchical chains of communication, or a hierarchically decomposed engineering system with hierarchical specification. Indeed, these two representations are synergistic, in that hierarchical organizations are generally the mechanism by which hierarchically decomposed systems are generated. The difficulty with this architecture is that there is a bandwidth limitation in the communication channels. The channels of communication pass through individual components. If we assume that each component has a limit on its variety, then we see that the communication channels are limited by the variety of their components.

This is a severe limitation on the variety of the system behavior, because in a more networked structure it is possible for the components at the bottom of the hierarchy to coordinate with each other directly in a way that would dramatically increase the variety of possible pairwise actions well above what can be coordinated through the hierarchy. This illustrates the well known phenomenon in engineering of the explosion of interface specification, and the dramatic efforts that are devoted to coordination of components. Indeed, the point is that while in the conventional decomposition strategy it is the components that are presumed to be the entities that require engineering, when systems become highly complex it is the coordination that requires the effort of engineering. Then the conventional strategy breaks down and other mechanisms are necessary. The formal proof of this statement requires one subtlety, which is quantifying the coordination above the level of behavior of an individual. The variety that is most limiting for a hierarchical organization is variety on a scale that requires more than one individual to perform a task, but is significantly below the number of individuals that form the system. It is the existence of large varieties at these intermediate scales that is not possible for hierarchical organizations. Either a completely independent or a completely dependent organizational behavior can be readily achieved. We describe this formalism in several steps.

Quantitatively, the understanding of the requirements of variety was articulated in Ashby's Law of Requisite Variety. Recently this law has been generalized to consider the issue of scale as well as variety. In the generalization it is assumed that the system is composed of a number of components, and that these components can be combined to perform specific tasks that might require more than a single component to perform.

More specifically, we assume that the responding system is composed of a number of subsystems, N, that are variously coordinated to respond to external contexts. The number of possible actions that the system can take, M, is not more than m^N, the product of the possible actions of each part, m. We could directly apply the Law of Requisite Variety for that case, but we further constrain the problem of effective function by assuming that effective actions require a sufficient variety at each scale of action corresponding to the requirements for action at that scale. At every scale, the variety of the system must be larger than the variety necessary for the task. It is conventional to measure variety, like information, in logarithmic units so that the total variety of a set of independent components $V = \log(M)$ is the sum of the variety of the components, $V = Nv$, where $v = \log(m)$. If we assume a simple coordination mechanism so that the system is partitioned into groups that are fully coordinated and that different groups are independent of each other, then the variety of actions of each group is the same as the variety of actions of any individual of that group, and the scale of action is just the number of individuals in that group. For the entire system the variety at scale k is $D(k) = vn(k)$ where $n(k)$ is the number of different k-member fully coordinated groups needed to perform the entire task, which therefore at a minimum requires $N = \sum kn(k)$ components to perform. The total variety of the task is proportional to the total number of subsets of any scale $V = \sum D(k)$.

With these assumptions, given a predetermined number of components N, the system can, in the extreme, perform a task of scale N, with variety equal to that of one component, or a task of scale one with variety N times as great. More generally the equation (obtained from $N = \sum kn(k)$)

$$Nv = \sum kD(k) \tag{1}$$

can be considered a constraint on the possible behavior patterns (sum rule) of a system due to different mechanisms of organization. It is often convenient to think about the variety of a system, $V(k)$, that has a scale k or larger, as this is the set of possible actions that can have at least that scale,

$$V(k) = \sum_{k'=k}^{N} D(k') \tag{2}$$

Then the total variety of the system is $V(1)$, and the sum rule can be written as:

$$\sum_{k=1}^{N} V(k) = Nv \tag{3}$$

The sum rule given by equation (1) or (3) describes the existence of a tradeoff between variety at different scales. Increasing the variety at one scale by changing the organizational form must come at the expense of variety at other scales. Our generalization of the Law of Requisite Variety is directly relevant to the analysis of coordination mechanisms of a biological or social organization. Specifically, it tells us how such coordination mechanisms are well or ill suited to the tasks being performed. Given the constraint imposed by the number of components, a successful organization has a coordination mechanism that ensures that the groups are coordinated at the relevant scale of tasks to be performed. This simple and intuitive statement is captured by the multiscale version of the Law of Requisite Variety.

A key issue is the concept of a hierarchical organization, of systems we build or of human organizations that build them. In considering the requirements of multiscale variety, we can state that in order for a system to be effective, it must be able to coordinate the right number of components to serve each task, while allowing the independence of other sets of components to perform their respective tasks without binding the actions of one such set to another. This now serves as a key characterization of system organization. Specifically, the Multiscale Law of Requisite Variety implies that in order for a system to be successful its coordination mechanisms must allow independence and dependence between components so as to allow the right number of sets of components at each scale.

How do we describe a coordinator / manager? A manager specifies the state of the subordinates and a coordination mechanism. We assume that at any particular time the manager can only coordinate a particular subset, indexed by w, of the subordinates, and at that time these subordinates are fully coordinated, while the others act independently (one cannot be in two places at the same time). $q(w)$ is the number of subordinates that are being coordinated, which, for values of zero or one corresponds to no coordination. A specification of the manager at a particular time thus can be written (s_m,w), where the state of s_m specifies the states of all the coordinated subordinates, while w specifies which subordinates are coordinated. For simplicity we do not count the redundancy provided by the manager (who we assume does not do the action only specifies it) and therefore s_m is not needed in the description of the system since it is redundant to the actions of the subordinates. We also neglect the information in specifying w by treating the information as conditional on the coordination mechanism. These assumptions can be relaxed without changing the conclusions. Then we have the multiscale variety for a particular coordination state given by:

$$D(k \mid w) = v(N - q(w))\delta_{k,1} + v\,\delta_{k,q(w)} \tag{4}$$

Combining coordination states, each with a probability $P(w)$ we have:

$$D(k) = \sum_{w} P(w)\delta_{q(w),k}v + \delta_{k,1}\sum_{w} P(w)(N - q(w))v \tag{5}$$

This gives the expected bound on the total coordination:

$$V(2) = \sum_{k=2}^{N} D(k) = \sum_{k=2}^{N}\sum_{w} P(w)\delta_{q(w),k}v \leq v \tag{6}$$

The inequality is the quite reasonable statement that the variety of the system for scales larger than one individual cannot be greater than the variety of the manager.

This coordination limitation is recursively applied to each level of managers for the set of individuals under their supervision so that the mutual information between individuals (workers or managers) at one level of organization is limited by the manager that supervises them. This implies, for example, that the combined mutual information between all workers is no more than the variety of the first level supervisors. Assuming that the variety of a manager is typically no more than the variety of a worker, we would expect that the limit of mutual information to be N/B where B is the branching ratio, i.e. the number of workers supervised by a single manager. Higher level managers are similarly restricted in their ability to coordinate the managers at the lower level. We note that in a conventional hierarchy when an upper level manager coordinates parts of the organization, this information must be communicated through the lower level managers. This also reduces the degree to which their own inter-worker coordination can be performed (i.e. to the extent that the higher level manager performs coordination, this reduces the capacity of the lower level managers to coordinate).

We can make a more direct connection to multiscale variety if we consider a somewhat generalized version of hierarchical control. In the generalized version of the hierarchy managers exist at a certain level of authority, supervising a certain fraction of the organization, but do not have a particular set of subordinates that they supervise (the "matrix organization" [20] is an intermediate case). By not including the constraint of a strict hierarchy that a manager has a particular subset of the individuals and cannot coordinate others outside of this subset we obtain an upper bound on the coordination of a more conventional hierarchy. If we include this additional constraint, then the coordination of the system is further limited since even only two individuals that are in different divisions of the organization require coordination by the CEO. For the generalized hierarchical model, we can generalize the equations above and reach a conclusion that

$$V(2)=\sum_{k=2}^{N} D(k) \le Cv \tag{7}$$

Where C is the number of managers. This states quite reasonably that the total variety of actions greater than the scale of one individual is not greater than the total variety of the managers. For managers having a certain limit on how many subordinates they can control, so that managers at level l can coordinate up to B^l subordinates, we further limit the number of those coordinated at larger scales by

$$V(B^{l-1}+1)=\sum_{k=B^{l-1}+1}^{N} D(k) \le \sum_{l'\ge l} C_{l'}v \tag{8}$$

which reasonably states that the variety of behaviors associated with a number of individuals is only as great as the variety of the managers that can coordinate that number of individuals.

For example, we consider the role of the CEO and assign him/her the obligation of determining those issues that are of relevance to the actions of a large proportion of individuals that are part of the organization. If we consider 10% to be the threshold fraction, then all decisions involving 10% of the individuals of the organization are coordinated by the CEO. The maximal possible variety of such portions (at this scale of action) is ten times the variety of a single individual. However, this cannot be done when coordinated by a single individual, as the maximum is the CEO's variety. More generally, we can categorically state that, to the extent that a single individual is coordinating the behavior of an organization, to that extent the coordination defined by mutual information cannot have a higher variety than an individual.

We see that for a hierarchically coordinated system the combined conditional mutual information of subunits of a manager cannot be greater than the variety of that manager. This is not a problem for either of two cases (dictated by environmental conditions): if the system has a simple coherent behavior, or if the manager exercises very little control so that the workers are almost totally independent of each other. It is a problem, however when the behaviors of subunits themselves have a high variety (greater than that of an individual) and must be coordinated. Thus, a hierarchical control system is well designed for relatively simple large scale behaviors, or for systems with very distributed control, but not for highly coordinated behaviors, i.e. when the coordination of these behaviors is more complex than a human being can communicate.

The recognition that hierarchical control is limited in its ability to coordinate was articulated for free market systems sixty years ago.[21] This limitation has also been recognized as relevant to the management of individual corporations based upon an understanding of human information processing rather than communication.[22,23] However, to our knowledge this is the first time that the limitations of hierarchical control have been formally demonstrated.[10] The demonstration required the representation of mutual information between multiple components described by multiscale variety.[1,10-13] This approach also demonstrates the limitations on capabilities of systems that rely upon individuals as liaisons between corporate divisions. [20]

There is one form of hierarchical control that is not ruled out by our discussions. When the set of possibilities is only a few, even if they are radically different from each other (involving changes in the action of many individuals), then the coordination/decision can be made by a single individual. This implies that that aspect of the organization is coherent, i.e., large scale and not of high variety. For example, the choice of whether or not to go to war can be made by an individual with only two possible decision states. However, this reflects the assumption that all aspects of the internal coordination necessary for the two states are made by others. Although this aspect of central control is not limited by our discussion, it is important to recognize the applicability of limitations by other arguments: the availability of the necessary information [21] and information processing to make the decision [22]. This information is related to the structure of the decision making process. The process must be able to contain prototypes of conditions and pair them with actions (or conditions and actions with effects).

We can consider these concepts from a phenomenological point of view. Centralized coordination of components was characteristic of scientific management as applied to the economy of the USSR that specified the coordination of industrial enterprises. Failures of this system in providing agricultural products of appropriate quantity but possibly more importantly of sufficient variety [1,24] led Gorbachev, First Deputy Prime Minster in charge of agriculture before becoming General Secretary of the Central Committee of the Communist Party, to institute reforms that preceded the collapse of the Soviet Union.

In summary, a generalization of the Law of Requisite Variety suggests that the effectiveness of a system organization can be evaluated by its variety at each scale of tasks to be performed. In its simplest form, when a system has a high degree of coordination then it is large scale. When it is not coordinated, allowing for independent component action, then it has high variety. The tradeoff of large scale action, as compared to the variety possible when actions of components are independent provides a direct analysis of system organization. While it does not specify that a particular system is capable of performing a task, it can provide a necessary condition for such effectiveness. In considering biological and social systems, such analysis provides a way of classifying their behavior and considering the functional role they play in survival and societal function. [1,5-10]

3 Enlightened Evolutionary Engineering

In the conventional systems engineering approach the project is recursively broken into subparts. The parts are then put together, with the task of selecting and coordinating the subprojects the domain of the systems engineer. The failure rate of such engineering projects in recent years has been remarkably high, costing many billions of dollars. [1-4]

The traditional approach to large engineering projects follows the paradigm established by the Manhattan project and the Space program. There are several assumptions inherent to this paradigm. First, that substantially new technology will be used. Second, the new technology to be used is based upon a clear understanding of the basic principles or equations that govern the system (i.e. the relationship between energy and mass, $E=mc^2$, for the Manhattan project, or Newton's laws of mechanics and gravitation $F=-GMm/r^2$ for the space program). Third, that the goal of the project and its more specific objectives and specifications are clearly understood. Fourth, that based upon these specifications, a design will be created essentially from scratch and this design will be implemented and, consequently the mission will be accomplished.

Large engineering projects today generally continue to follow this paradigm. Projects are driven by a need to replace old "obsolete" systems with new systems, and particularly to use new technology. The time line of the project involves a sequence of stages: a planning stage at the beginning, giving way to a specification stage, a design stage, and an implementation stage. The various stages of the process all assume that managers know what needs to be done and that this information can be included in a specification. Managers are deemed successful or unsuccessful depending on whether this specification is achieved. On the technical side, modern large engineering projects generally involve the integration of systems to create larger systems. Their goals include adding multiple functions that have not been possible before, and they are expected to satisfy additional constraints, especially constraints of reliability, safety and security.

The images of success in the Manhattan and Space Projects remain with us. What really happens with most large engineering projects is much less satisfactory. Many projects end up as failed and abandoned. This is true despite the tremendous investments that are made. The largest documented financial cost for a single project, the Federal Aviation Administration (FAA) Advanced Automation System was the government effort to improve air traffic control in the United States. Many of the major difficulties with air traffic delays and other limitations are blamed on the antiquated / obsolete air traffic control system. This system, originally built in the 1950s, used remarkably obsolete technology, including 1960s mainframe computers and equipment based upon vacuum tubes [25], with functional limitations that would compel any modern engineer into laughter. Still, an effort that cost $3-6 billion between 1982 and 1994 was abandoned without improving the system. While the failure of government projects are frequently blamed on specific issues related to government acquisition, a general survey of large software engineering projects in 1995 by the Standish Group International [14] showed that such failures were widespread in both private and public sector projects. This study classified projects according to whether they met the stated goals of the project, the time table, and cost estimates. They found that under 20% of the projects were on-time, on-budget and on-function (projects at large companies had a lower rate of under 10% success), over 50% of the projects were "challenged" which meant they were over budget, typically by a factor of two, they were over schedule by a factor of two, and did not meet about two-thirds of the original functional specifications. The remaining 30% of the projects were called "impaired" which meant that they were abandoned. When considering the major investments of time and money these projects represent, the numbers are staggering, easily reaching $100 billion each year in direct costs. The high percentage of failures and the remarkable percentage of challenged projects suggest that there is a systematic reason for the difficulty involved in large engineering projects beyond the specific reasons for failure that one might identify in any one case.

Indeed, despite various efforts to improve acquisition of large systems, successors of the Advanced Automation System that are being worked on today are finding the going slow and progress limited [26]. From 1995 until today, major achievements include replacing mainframe computers, replacing communications switching system, and the en-route controller radar stations. The replacement of the Automated Radar Terminal System at Terminal Radar Facilities responsible for air traffic control near airports (the Standard Terminal Automation Replacement System (STARS) program), faced many of the problems that affected the Advanced Automation System: cost overruns, delays, and safety vetoes of implementation, and was implemented in 2002 by FAA emergency decree. Still, the new equipment continues to be used in a manner that follows original protocols used for the old equipment.

A fundamental reason for the difficulties with modern large engineering projects is their inherent complexity. Complexity is generally a characteristic of large engineering projects today. Complexity implies that different parts of the system are interdependent so that changes in one part may have effects on other parts of the system. Complexity may cause unanticipated effects that lead to failures of the system. These “indirect” effects can be discussed in terms of multiple feedback loops among portions of the system, and in terms of emergent collective behaviors of the system as a whole [1,5]. Such behaviors are generally difficult to anticipate and understand. Despite the superficial complexity of the Manhattan and Space Projects, the tasks that they were striving to achieve were relatively simple compared to the problem of air traffic control. To understand complexity of air traffic control it is necessary to consider the problem of 3-dimensional trajectory separation --- ensuring the paths of any two planes do not intersect at the same time; the many airplanes taking off and landing in a short period of time; and the remarkably low probability of failure that safety constraints impose. Failure in any one case may appear to have a specific cause, but the common inability to implement high cost systems can be attributed to their intrinsic complexity.

While the complexity of engineering projects has been increasing, it is important to recognize that complexity is not new. Indeed, engineers and managers are generally aware of the complexity of these projects and have developed systematic techniques to address them. There are several strategies that are commonly used including modularity, abstraction, hierarchy and layering. These methods are useful, but at some degree of interdependence they become ineffective. Modularity is a well recognized way to separate a large system into parts that can be individually designed and modified. However, modularity incorrectly assumes that a complex system behavior can be reduced to the sum of its parts. As systems become more complex the design of interfaces between parts occupies increasing attention and eventually the process breaks down. Abstraction simplifies the description or specification of the system. However abstraction assumes that the details to be provided to one part of the system (module) can be designed independently of details in other parts. Modularity and abstraction are generalized by various forms of hierarchical and layered specification, whether through the structure of the system, or through the attributes of parts of a system (e.g. in object oriented programming). Again, these two approaches either incorrectly portray performance or behavioral relationships between the system parts or assume details can be provided at a later stage. Similarly, management has developed ways to coordinate teams of people working on the same project through various carefully specified coordination mechanisms.

One way to address the difficulty of complex projects is to simplify what is attempted. However, simplifying the function of an engineered system is not always possible because the necessary or desired core function is itself highly complex. When the inherent nature of a complex task is too large to deal with using conventional large engineering processes, a better solution is to use an evolutionary process [1-3] to create an environment in which continuous innovation can occur.

Evolutionary processes, commonly understood to be analogous to free market competition, are based on incremental iterative change. However, there are basic differences between evolution and the notion of incremental engineering. Among these is that evolution assumes that many different systems exist at the same time, and that changes occur to these systems in parallel. The parallel testing of many different changes that can be combined later is distinctly different from conventional incremental engineering. The use of parallel initial exploration has been advocated in engineering [27]. However, this approach is also unlike evolution, because it leads to the selection of a single option rather than multiple parallel implementation. Multiple parallel implementation is more similar to the parallel and largely independent exploration of product improvements by different companies in a market economy, especially when there are many small companies. Another basic idea of evolution is that much testing is done "in the field"; the process of learning about effective solutions occurs through direct feedback from the environment. There are many more aspects of evolution that should be understood in order to make effective use of this process in complex large engineering projects. Even the conventional concepts of evolution as they are currently taught in basic biology courses are not sufficient to capture the richness of modern ideas about evolution [5 (ch. 6),28-30].

Many of the more recent programming strategies, e.g. spiral development, extreme programming, and the open source movement, embody features of evolutionary processes. Still, a better understanding is necessary in order to realize the promise of evolutionary methods. The objective revolves around mimicry of the processes that promote rapid innovation through competition. The creation of an effective "artificial ecology" or "artificial economy" requires design. In and of itself, a competitive system is not self-sustaining as it tends to become stuck through monopolization or self-destructive behavior.

To introduce the concepts of evolution it is helpful to start from the conventional perspective, then augment it with some of the modern modifications. Evolution is about the change in a population of organisms over time. This population changes not because the members of the population change directly, but because of a process of generational replacement by offspring that differ from their parents. The qualities of offspring are different from their parents, in part, because some parents have more offspring than others. The process by which the number of offspring are determined, termed selection, is considered a measure of organism effectiveness / fitness. Offspring tend to inherit traits of parents. Traits are modified by sexual reproduction and mutations that introduce novelty / variation. This novelty allows progressive changes over many generations. Thus, in the conventional perspective evolution is a process of replication with variation followed by selection based upon competition. In contrast with an engineering view where the process of innovation occurs through concept, design, specification, implementation and large scale manufacture, the evolutionary perspective would suggest that we consider the population of functioning products that are in use at a particular time as the changing population that will be replaced by new products over time. The change in this population occurs through the selection of which products increase their proportion in the population. This process of evolution involves the decisions of people as well as the changes that occur in the equipment itself.

It may be helpful to point out that this approach (the treatment of the population of engineered products as evolving) is quite different than the approach previously used to introduce evolution in an engineering context through genetic algorithms or evolutionary programming (GA/EA) [31,32]. The GA/EA approach has considered automating the process of design by transferring the entire problem into a computer. According to this strategy, we develop a representation of possible systems, specify the utility function, implement selection and replication and subsequently create the system design in the computer. While the GA/EA approach can help in specific cases, it is well known that evolution from scratch is slow. Thus it is helpful to take advantage of the capability of human beings to contribute to the design of systems. The objective of the use of evolutionary process described here is to avoid relying upon an individual human being to design systems that can perform highly complex tasks. A computer by itself cannot solve such problems either. Our objective here is to embed the process of design into that of many human beings (using computers) coordinated through an evolutionary process.

The basic concept of designing an evolutionary process is to create an environment in which a process of innovation and creative change takes place. To do this we develop the perspective that tasks to be performed are analogous to resources in biology. Individual parts of the system, whether they are hardware, software or people involved in executing the tasks are analogous to various organisms that are involved in an evolutionary process. Changes in the individual parts take place through introducing alternate components (equipment, software, training or by moving people to different tasks). All of these changes are part of the dynamics of the system. Within this environment it is possible for conventional engineering of equipment or software components to occur. The focus of such engineering efforts is on change to small parts of the system rather than on change to the system as a whole. This concept of incremental replacement of components (equipment, software, training, tasks) involves changes in one part of the system, not in every part of the system. Even when the same component exists in many parts of the system, changes are not imposed on all of these parts at the same time. Multiple small teams are involved in design and implementation of these changes. It is important to note that this is the opposite of standardization—it is the explicit imposition of variety. The development environment should be constructed so that exploration of possibilities can be accomplished in a rapid (efficient) manner. Wider adoption of a particular change, corresponding to reproduction in biology, occurs when experience with a component indicates improved performance. Wider adoption occurs through informed selection by individuals involved. This process of "selection" explicitly entails feedback about aggregate system performance in the context of real world tasks.

Thus the process of innovation involves multiple variants of equipment, software, training or human roles that perform similar tasks in parallel. The appearance of redundancy and parallelism is counter to the conventional engineering approach which assumes specific function assignments rather than parallel ones. This is the primary difference between evolutionary processes and incremental approaches to engineering. The process of overall change consisting of an innovation that, for example, replaces one version of a particular type of equipment with another, occurs in several stages. In the first stage a new variant of the equipment (or other component) is introduced. Locally, this variant may perform better or worse than others. However, overall, the first introduction of the equipment does not significantly affect the performance of the entire system because other equipment is operating in parallel. The second stage occurs if the new variant is more effective: others may adopt it in other parts of the system. As adoption occurs there is a load transfer from older versions to the new version in the context of competition, both in the local context and in the larger context of the entire system. The third stage involves keeping older systems around for longer than they are needed, using them for a smaller and smaller part of the load until eventually they are discarded 'naturally'. Following a single process of innovation, is, however, not really the point of the evolutionary engineering process. Instead, the key is recognizing the variety of possibilities and subsystems that exist at any one time and how they act together in the process of innovation.

The conventional development process currently used in large engineering projects is not entirely abandoned in the evolutionary context. Instead, it is placed within a larger context of an evolutionary process. This means that individuals or teams that are developing parts of the system can still use well known and tested strategies for planning, specification, design, implementation and testing. The important caveat to be made here is that these tools are limited to parts of the system whose complexity is appropriate to the tool in use. Also, the time scale of the conventional development process is matched to the time scale of the larger evolutionary process so that field testing can provide direct feedback on effectiveness. This is similar to various proposals suggested for incremental iterative engineering. What is different is the importance of parallel execution of components in a context designed for redundancy and robustness, so that the implementation of alternatives can be done in parallel and effective improvements can be combined. At the same time, the ongoing variety provides robustness to changes in the function of the system. Specifically, if the function of the system is changed because of external changes, the system can adapt rapidly because there are many possible variants of subsystems that can be employed.

Understanding a complex system approach to design and implementation involves recognizing the many differences between the natural evolutionary process and traditional engineering practices. Enlightened Evolutionary Engineering (E^3) employs, among others, the following key concepts, that may be contrasted to traditional engineering practices.

Focus on Creating an Environment and Process Rather Than a Product

Ongoing change in a system is the underlying mechanism of creation, not the formulation and execution of plans. Encouraging and safeguarding this ongoing change and monitoring its outcomes are the absolute essentials of an evolutionary-based process.

Continually Build on What Already Exists

Off-line engineering of complex systems is impractical because the complexities of their environment and true functional requirements do not permit practical specification or testing prior to implementation. In complex systems, correct expectations and testing both depend on the immediate consequences of current operations.

Individual Components Must Be Modifiable in situ

The interdependencies between system components must be such that individual components can be modified in situ. In practice this requires the following point.

Operational Systems Include Multiple Versions of Functional Components

Complex systems should be understood as populations rather than as rigid assemblies of unique components. Individual components can overlap substantially in terms of both functionality and interaction. Evolutionary processes impact both populations and individuals. Redundancies are not always unwanted inefficiencies.

Utilize Multiple Parallel Development Processes

The existence of populations of components allows multiple parallel efforts to explore modifications that might (but that are not guaranteed) to improve system components and/or total system capability.

Evaluate Experimentally In-situ

Testing and experimentation increasingly overlap. Off-line qualification testing becomes a prelude to active field testing for components in a large variety of operational environments. Results (including unexpected results) are ratified or rejected as they occur based on then-current overall system capability.

Increase Utilization of More Effective Components, Gradually

The replacement of components cannot be abrupt as testing is never complete and operation is continuous. Augmentation and parallel operation is the preferred approach.

Effective Solutions to Specific Problems Cannot Be Anticipated

Specification efforts cannot assume that the most efficient or effective solutions can be anticipated in advance of an exploration and discovery process involving multiple parallel development efforts. Such an assumption is invalid, and is increasingly seen to be so the more complex any solution must be to even marginally succeed. Moreover, this assumption remains false no matter how long a problem is worked and progressively better solutions are found.

The “Integration” of Complex Systems

In order to operate a E^3 process, the concept of integration must be radically rethought. A systematic and effective application of the ideas in this paper involves a “paradigm shift” from “complete system specification” to the creation of environments that are conducive to ongoing change in components of systems while supporting the more or less constant evaluation of their overall effectiveness through virtual as well as real world testing.

4 Conclusions

It is important to appreciate that there are fundamental reasons that highly trained systems engineers have been unable to successfully complete highly complex engineering projects in recent years. Extending the existing decomposition based approach will not solve these problems. The application of multiscale analysis reveals that the coordination between components that is required to develop such systems is incompatible with decomposition. This can be most easily understood as an underlying bandwidth limitation in the hierarchical structure in which the decomposition of the design is performed.

The solution to this problem is to develop an environment for parallel design teams to develop components that can be field tested and compete for wider adoption. This approach underlies both the creation of complex biological systems and many complex social system through the process of market competition.

References

1. Y. Bar-Yam, Making Things Work; Solving complex problems in a complex world, (NECSI Knowledge Press, 2004).
2. D. Braha, A. Minai, and Y. Bar-Yam, eds. Engineered Complex Systems, NECSI Knowledge Press, 2004.
3. Y. Bar-Yam, Enlightened Evolutionary Engineering / Implementation of Innovation in FORCEnet, Report to Chief of Naval Operations Strategic Studies Group, 2002 (Brief 2000).
4. Y. Bar-Yam, When Systems Engineering Fails --- Toward Complex Systems Engineering, International Conference on Systems, Man & Cybernetics, 2003, Vol. 2, 2021- 2028, IEEE Press, Piscataway, NJ, 2003.
5. D. Braha and O. Maimon, A Mathematical Theory of Design: Foundations, Algorithms and Applications, Kluwer, Boston, 1998.
6. Y. Bar-Yam, Dynamics of Complex Systems, (Perseus, Reading, MA, 1997).
7. Y. Bar-Yam: General Features of Complex Systems, in Encyclopedia of Life Support Systems (EOLSS), UNESCO, EOLSS Publishers, Oxford ,UK, 2002
8. Y. Bar-Yam: Complexity rising: From human beings to human civilization, a complexity profile, in Encyclopedia of Life Support Systems (EOLSS), UNESCO, EOLSS Publishers, Oxford ,UK, 2002
9. Y. Bar-Yam, "Unifying Principles in Complex Systems" in Converging Technology (NBIC) for Improving Human Performance, M. C. Roco and W. S. Bainbridge, Dds., Kluwer, 2003.
10. Y. Bar-Yam, Multiscale Variety in Complex Systems, Complexity 9:4, pp. 37-45, 2004.
11. Y. Bar-Yam, A Mathematical Theory of Strong Emergence using Multiscale Variety, Complexity 9:6, pp. 15-24, 2004.
12. Y. Bar-Yam: Multiscale Complexity / Entropy, Advances in Complex Systems 7, pp. 47-63, 2004.
13. S. Gheorghiu-Svirschevski and Y. Bar-Yam, Multiscale analysis of information correlations in an infinite-range, ferromagnetic Ising system, Phys.Rev. E 70, 066115 (2004)
14. Standish Group International, The CHAOS Report, 1994.
15. G. S. Lynn, J. G. Morone and A. S. Paulson, "Marketing and discontinuous innovation: The probe-and-learn process," California Management Rev., Vol. 38, No. 3, pp. 8–36, 1996.
16. R. W. Veryzer, "Discontinuous innovation and the new product development process," J. Product Innovation Management, Vol. 15, pp. 304–321, 1998

17. DoD Directive 5000.1, “The Defense Acquisition System,” May 12, 2003.
18. See e.g. Agile Software Development, CrossTalk, Vol. 15, No. 10, Oct. 2002.
19. M. T. Pich, C. H. Loch and A. De Meyer, “On Uncertainty, Ambiguity and Complexity in Project Management” Management Science 48, 1008-1023, 2002.
20. J. Galbraith, Designing Complex Organizations, (Addison-Wesley, 1973)
21. F. A. Hayek, The Road to Serfdom, (Routledge, 1944)
22. J. G. March and H. A. Simon, Organizations; 2nd Ed., (John Wiley & Sons: New York, 1958)
23. H. Mintzberg, The Structuring of Organizations (Prentice-Hall, 1979)
24. I. Birman, Personal Consumption in the USSR and the USA (Palgrave Macmillan, 1989).
25. Committee on Transportation and Infrastructure Computer Outages at the Federal Aviation Administration's Air Traffic Control Center in Aurora, Illinois [Field Hearing in Aurora, Illinois] hpw104-32.000 HEARING DATE: 09/26/1995.
26. U.S. House Committee on Transportation and Infra-structure, FAA Criticized For Continued Delays In Modernization Of Air Traffic Control System, Mar. 14, 2001.
27. D. K. Sobek, A. C. Ward and J. K. Liker, “Toyota’s principles of set-based concurrent engineering,” Sloan Management Rev., Vol. 40, pp. 67–83, 1999.
28. E. Rauch, H. Sayama and Y. Bar-Yam, “The role of time scale in fitness,” Phys. Rev. Lett. 88, 228101-4 (2002).
29. J. K. Werfel and Y. Bar-Yam, The evolution of reproductive restraint through social communication, PNAS 101, 11019-11024 (2004).
30. Y. Bar-Yam: Formalizing the gene centered view of evolution, Advances in Complex Systems 2, 277-281 (1999).
31. L. J. Fogel, A. J. Owens and M. J. Walsh, Artificial Intelligence through Simulated Evolution, Wiley, New York, 1966
32. J. H. Holland, Adaptation in Natural and Artificial Systems, 2d ed. MIT Press, Cambridge, 1992.

Adaptive Information Infrastructures for the e-Society

Mihaela Ulieru

Electrical and Computer Engineering Department,
The University of Calgary,
2500 University Dr. NW,
Calgary, Alberta, T1N 1N4 Canada
ulieru@ucalgary.ca
http://www.enel.ucalgary.ca/People/Ulieru/

Abstract. Positioned at the confluence between human/machine and hardware/software integration and backed by a solid proof of concept realized through several scenarios encompassing e-Securities, e-Health, and e-Logistics for global manufacturing and emergency response management, this work exploits latest advances in information and networking technologies to set a systematic framework for the design of the information infrastructures (coined as AIIs - Adaptive Information Infrastructures) destined to fuel tomorrow's e-Society. Designed following the natural laws of evolution, which merge self-organization and natural selection [1], these socially embedded information infrastructures can adapt to fulfill various needs as their environment demands. Computational intelligence techniques endow the AIIs with learning and discovery capabilities, emulating social and biological behavior. AIIs are destined to become an integral part of our life by supporting, rather than disturbing, a framework that facilitates strategic partnerships while providing greater user-friendliness, more efficient services support, user-empowerment, and support for human interactions.

Keywords: Distributed artificial intelligence, information infrastructures, emergency response management, e-Health, Cybersecurities, emergence, self-organization, evolution.

1 Rationale

Today's electronic information technologies are linking our world, enabling partnerships otherwise impossible in all areas of our life. From e-Commerce and e-Business to e-Learning and e-Health the economic strategies as well as the routine professional practices have been irreversibly contaminated with the spice of electronic connectivity. Supported by this technological leverage, new paradigms have emerged with models that are dynamic, autonomous, self-organizing and proactive, generically coined as 'intelligent'. In particular Multi-Agent Systems (MAS) have changed the software world, and with it the world of information technologies. With the reasoning encapsulated in *societies of software agents*, having a life of their own in Cyberspace, the Internet becomes a *dynamic environment* through which agents move from place to place to deliver their services and eventually to compose them with the ones of other agents, just like people cooperate, by exchanging services and/or putting together their competencies in a larger, more complex service.

In today's dramatic context there is an acute need for such new techniques capable to deal with critical aspects such as emergency response management, network, information and national security enhancement, population health and quality of life improvement, etc. To meet this need we propose a systematic approach to the design and implementation of such dynamic environments supporting coalition formation, which we refer to as *adaptive information infrastructures* (AII). AIIs could glue together the best organizations capable to cooperate in the timely solving of a crisis, and support the coordination of activities across such an extended cooperative organization, getting clarity to emerge from the fog of information and help make the best decisions out of the crisis chaos.

To influence the development of this technology in a human-friendly way our approach builds on the natural laws/patterns of self-organization according to which adaptive / intelligent systems emerged in the process of universes' evolution [7]. Our approach [8] addresses this by enabling information infrastructures for various applications. For example, for global production integration [9], we developed a methodology for dynamic resource management and allocation across distributed (manufacturing) organizations [10], [11]. The approach integrates multi-agent technology with the *holonic* paradigm proposed by A. Koestler in his attempt to create a model for self-organization in biological systems [12].

2 Holonics

Koestler postulated a set of underlying principles to explain the self-organizing tendencies of social and biological systems. He proposed the term *holon* to describe the elements of these systems. This term is a combination of the Greek word *holos*, meaning "whole", with the suffix *-on* meaning "part", as in prot*on* or neur*on*. This term reflects the tendencies of holons to act as autonomous entities, yet cooperating to form apparently self-organizing hierarchies of subsystems, such as the cell/tissue/organ/system hierarchy in biology.

Starting from the empirical observation that, from the Solar System to the Atom the Universe is organized into self-replicating structures of nested hierarchies intrinsically embedded in the functionality of natural systems, in his attempt to creating a model for self-organization in biological systems, Koestler has identified structural patterns of self-replicating structures, named holarchies. Holarchies have been envisioned as models for the Universe's self-organizing structure in which holons at several levels of resolution in the nested hierarchy [13] (Fig. 1) behave as autonomous wholes and yet as cooperative parts for achieving the goal of the holarchy.

In such a nested hierarchy each holon is a sub-system retaining the characteristic attributes of the whole system. What actually defines a holarchy is a *purpose* around which holons are clustered and subdivided in sub-holons at several levels of resolution according to the organizational dissectibility required. A Confederation is a *political holarchy*, for example having Canada at the highest level of resolution then the provinces at the immediate lower level, and finally the cities at the lowest levels in the hierarchy. Each individual person is regarded as a primitive holon within this social holarchy. Other examples are: a global enterprise is a collaborative purpose- driven/

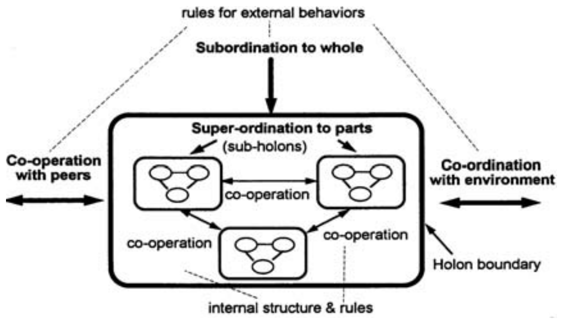

rules for external behaviors
Subordination to whole
Super-ordination to parts
(sub-holons)
Co-operation with peers
Co-ordination with environment
co-operation
co-operation
co-operation
Holon boundary
internal structure & rules

Fig. 1. Generic Model of a Holarchy

market-driven holarchy; a distributed manufacturing system is a production-driven holarchy, the organism is a survival-driven holarchy. The Universe is an *evolution*-driven holarchy.

3 A Mathematics of Emergence

In his seminal book [1] Stuart Kaufmann postulates that life emerged in the Universe through collective autocatalytic processes fueled by self-organization and natural selection. As result of the process of evolution driven by power laws and autocatalicity, emergence endows the dynamics of composite systems with properties unidentifiable in their individual parts. The phenomenon of emergence involves on one side *self-organization* of the dynamical systems such that the synergetic effects can occur and on the other side interaction with other systems from which the synergetic properties can *evolve* in a new context.

The flow of information and matter across a holonic organization defines several levels of granularity (Fig. 1) across which we emulate the mechanism of emergence to enable the dynamic creation, refinement and optimization of flexible ad-hoc AIIs as coordination backbones for the distributed organization, capable to bring together the best resources available (within reach) depending on the needs of the particular crisis to be addressed.

As such, the phenomenon of emergence involves two distinct steps, namely:

- *Self-organization* of the dynamical systems such that the synergetic effects can occur
- Interaction with other systems from which the synergetic properties can *evolve*

We integrate emergence into the holonic paradigm [15] to create, refine and optimize AIIs. Self-organization is achieved by minimizing the entropy measuring the fuzzy information spread across the multi-agent system [10]. This will cluster the resources (agents), ensuring interaction between the system's parts to reach its objectives timely, efficiently and effectively. Evolution is enabled by interaction with external systems (agents); for example, via a genetic search in cyberspace that mimics mating with most fit partners in natural evolution [16] or by means of dynamic discovery services [17]. In the sequel we present the essence of our formalism.

3.1 Self-Organization

A multi-agent system (MAS) is regarded as a dynamical system in which agents exchange information organized through reasoning into knowledge about the assigned goal [10]. Optimal knowledge corresponds to an optimal level of information organization and distribution among the agents. It seems natural to consider the *entropy* as a measure of the degree of order in the information spread across the multi-agent system [2]. This information is usually uncertain, requiring several ways of modeling to cope with the different aspects of the uncertainty. Fuzzy set theory offers an adequate framework that requires the use of generalized fuzzy entropy [3].

One can envision the agents in the MAS as being under the influence of an information "field" which drives the inter-agent interactions towards achieving "equilibrium" with other agents with respect to this entropy [10]. The generalized fuzzy entropy is the measure of the "potential" of this field and equilibrium for the agents under this influence corresponds to an optimal organization of the information across the MAS with respect to the assigned goal's achievement. When the goal of the MAS changes (due to unexpected events, such as need to change a peer, machine breakdown, etc.) the equilibrium point changes as well inducing new re-distribution of information among the agents with new emerging agent interactions. This mechanism enabling dynamic system re-configuration with re-distribution of priorities is the essence of the emergent dynamic holonic structure. In this section, we will prove that when the agents are clustering into a holonic structure the MAS reaches equilibrium, which ensures optimal accomplishment of the assigned goal (task).

A. Vagueness Modeling in MAS – The Problem

It is already well known that among the other uncertainty facets, *vagueness* deals with information that is *inconsistent* [4]. In the context of MAS, this means that the clear distinction between a possible plan reaching the imposed goal and a plan leading, on the contrary, to a very different goal is hardly distinguishable. We call *partition* the clustering configuration in which the union of all clusters is identical to the agent set when clusters are *not* overlapping. If the clusters overlap (i.e. some agents are simultaneously in two different clusters) the clustering configuration is called a *cover*. We define a *plan* as being the succession of all states through which the MAS transitions until it reaches its goal. Each state of the MAS is described by a certain clustering configuration covering the agents set. Starting from this uncertain information, the problem is to provide fuzzy models of MAS, useful in selecting *the least uncertain (the least vague) source-plan.*

B. Mathematical Formulation of the Problem

Denote by $\mathsf{A}_N = \{a_n\}_{n\in\overline{1,N}}$ the set of $N \geq 1$ agents that belong to the MAS. Based only on the initial uncertain information, one can build a family $\mathsf{P} = \{\mathsf{P}_k\}_{k\in\overline{1,K}}$, containing $K \geq 1$ collections of clustering configurations, for a preset global goal. Each P_k ($k \in \overline{1,K}$) can be referred to as a source-plan in the sense that it can be a source of partitions for a MAS plan. Thus, a source-plan is expressed as a collection of $M_k \geq 1$ different clustering configurations covering A_N, possible to occur during the MAS evolution towards its goal: $\mathsf{P}_k = \{P_{k,m}\}_{m\in\overline{1,M_k}}$. The only available information about P_k is the degree of occurrence associated to each of its configurations, $P_{k,m}$, which can be assigned as a number $\alpha_{k,m} \in [0,1]$. Thus, the corresponding degrees of occurrence are members of a two-dimension family $\{\alpha_{k,m}\}_{k\in\overline{1,K};m\in\overline{1,M_k}}$, which, as previously stated, quantifies all the available information about MAS.

In this framework, we aim to construct a measure of uncertainty, V (from "vagueness"), fuzzy-type, real-valued, defined on the set of all source-plans of A_N and optimize it in order to select the least vague source-plan from the family $\mathsf{P} = \{\mathsf{P}_k\}_{k\in\overline{1,K}}$:

$$\mathsf{P}_{k_0} = \arg\underset{k\in\overline{1,K}}{\underline{\text{opt}}}\, V(\mathsf{P}_k)\text{, } \textit{where } k_o \in \overline{1,K}. \tag{1}$$

The cost function V required in problem (1) will be constructed by using a *measure of fuzziness* [4]. We present hereafter the steps of this construction.

C. Constructing Fuzzy Relations Between Agents

We model agent interactions through fuzzy relations considering that two agents are in relation if they exchange information. As two agents exchanging information are as well in the same cluster one can describe the clustering configurations using these fuzzy relations. The family of fuzzy relations, $\{\mathsf{R}_k\}_{k\in\overline{1,K}}$, between the agents of MAS (A_N) is built using the numbers $\{\alpha_{k,m}\}_{k\in\overline{1,K};m\in\overline{1,M_k}}$ and the family of source-plans $\{\mathsf{P}_k\}_{k\in\overline{1,K}}$. Consider $k \in \overline{1,K}$ and $m \in \overline{1,M_k}$ arbitrarily fixed. In construction of the fuzzy relation R_k, one starts from the observation that associating agents in clusters is very similar to grouping them into *compatibility* or *equivalence classes*, given a (binary) crisp relation between them. The compatibility properties of reflexivity and symmetry are fulfilled for covers (overlapped clusters), whereas the equivalence conditions of compatibility and transitivity stand for partitions. The corresponding crisp relation denoted by $R_{k,m}$, can be described by the statement: *two agents are related if they belong to the same cluster.* The facts that a and b are, respectively are not in the relation $R_{k,m}$ (where $a,b\in \mathsf{A}_N$) are expressed by "$aR_{k,m}b$" and "$a\neg R_{k,m}b$". The relation $R_{k,m}$ can also be described by means of a $N\times N$ matrix $H_{k,m} \in \Re^{N\times N}$ - the *characteristic matrix* - with elements ($H_{k,m}[i,j]$) being only 0 or 1, depending on whether the agents are or not in the same cluster. (Here, $\Re$ is the real numbers set.) Thus:

$$H_{k,m}[i,j] \overset{def}{=} \begin{cases} 1 \text{ , } a_i R_{k,m} a_j \\ 0 \text{ , } a_i \neg R_{k,m} a_j \end{cases} \text{, } \forall i,j \in \overline{1,N}. \tag{2}$$

This matrix is symmetric (obviously, if $aR_{k,m}b$, then $bR_{k,m}a$) and with unitary diagonal (since every agent is in the same cluster with itself). It allows us to completely specify only the configuration $P_{k,m}$ (for the proof see [10].)

As such, the relation $R_{k,m}$ defined by the agents' inclusion in the same cluster is uniquely assigned to the clustering configuration $P_{k,m}$ (no other configuration can be described by $R_{k,m}$). Thus, each crisp relation $R_{k,m}$ can be uniquely associated to the degree of occurrence assigned to its configuration: $\alpha_{k,m}$. Together, they can define a so-called α*-sharp-cut* of the fuzzy relation R_k, by using the equality ($=$) instead of inequality ($\geq$) in the classical definition of α- *cut*. Therefore, the crisp relation $R_{k,m}$ is a α-sharp-cut of R_k, defined for $\alpha_{k,m}$.

Consequently, we can construct an elementary fuzzy (binary) relation $\mathsf{R}_{k,m}$ whose membership matrix is expressed as the product between the characteristic matrix $H_{k,m}$,defined by (2), and the degree of occurrence $\alpha_{k,m}$, that is: $\alpha_{k,m}H_{k,m}$. This fuzzy set of $\mathsf{A}_N \times \mathsf{A}_N$ is also uniquely associated to $P_{k,m}$.

If $k \in \overline{1,K}$ is kept fixed, but m varies in the range $\overline{1,M_k}$, then a family of fuzzy elementary relations is generated: $\{\mathsf{R}_{k,m}\}_{m\in\overline{1,M_k}}$. Naturally, R_k is then defined as the fuzzy union:

$$\mathsf{R}_k \overset{def}{=} \bigcup_{m=1}^{M_k} \mathsf{R}_{k,m} . \tag{3}$$

$$\mathsf{M}_k \overset{def}{=} \underset{m\in\overline{1,M_k}}{\underline{\max\bullet}}\left\{\alpha_{k,m}H_{k,m}\right\} \in \Re^{N\times N} , \tag{4}$$

where "$\max\bullet$"acts on matrix elements and not globally, on matrices. The equations (3) and (4) are very similar to the *resolution form* of R_k, as defined in [16]. Here however, some $\alpha_{k,m}$ (in general, with small values) can disappear from the membership grades of R_k.

Obviously, since all matrices $\alpha_{k,m}H_{k,m}$ are symmetric, M_k from (4) is symmetric as well, which means that R_k is a fuzzy symmetric relation. The fuzzy reflexivity is obvious (non-zero elements of main diagonal). Thus, R_k is at least a *proximity* relation. The manner in which the degrees of occurrence are assigned to partitions greatly affects the quality of the fuzzy relation. Although all its α-sharp-cuts could be equivalence relations, it is not necessary that the resulting fuzzy relation be a *similarity* one (i.e. fuzzy reflexive, symmetric and transitive). But it is at least a proximity relation, as explained above.

The fuzzy transitivity, expressed as follows:

$$\mathsf{M}_k \geq\bullet \left(\mathsf{M}_k \circ \mathsf{M}_k\right), \tag{5}$$

is the most difficult to ensure. Here "$\geq\bullet$" acts on matrix elements, and "$\circ$" denotes composition of the corresponding fuzzy relations. In case of *max-min transitivity*, this is expressed analogously to classical matrix multiplication, where the max operator is used instead of summation and min instead of product:

$$\mathsf{M}_k[i,j] \geq \left(\mathsf{M}_k \circ \mathsf{M}_k\right)[i,j] = \max_{n \in \overline{1,N}} \min\left\{\mathsf{M}_k[i,n], \mathsf{M}_k[n,j]\right\}, \tag{6}$$

where $i, j \in \overline{1,N}$ and $\mathsf{M}[i,j]$ is the current element of matrix M .

The equations (5) or (6) suggest an interesting procedure to construct similarity relations starting from proximity ones, by using the notion of *transitive closure*. A transitive closure of a fuzzy relation R is, by definition, the minimal transitive fuzzy relation that includes R . (Here, "minimal" is considered with respect to inclusion on fuzzy sets.)

So far, a bijective map (according to **Theorem 1**) between $\mathsf{P} = \{\mathsf{P}_k\}_{k \in \overline{1,K}}$ and $\mathsf{R} = \{\mathsf{R}_k\}_{k \in \overline{1,K}}$, say T , was constructed:

$$T(\mathsf{P}_k) = \mathsf{R}_k \ , \ \forall k \in \overline{1,K}. \tag{7}$$

D. The Measure of Fuzziness

The next step aims to construct a measure of fuzziness over the fuzzy relations on $\mathsf{A}_N \times \mathsf{A}_N$, that will be used to select the "minimally fuzzy" relation within the set $\mathsf{R} = \{\mathsf{R}_k\}_{k \in \overline{1,K}}$.

One important class consists of measures that evaluate "the fuzziness" of a fuzzy set by taking into consideration both the set and its (fuzzy) complement. From this large class, we have selected the *Shannon measure*, derived from the generalized Shannon's function:

$$\left[\begin{array}{l} S : [0,1]^M \quad \to \Re_+ \\ (x_1,\ldots,x_M) \mapsto S(x) \overset{def}{=} -\sum_{m=1}^{M}\left[x_m \log_2 x_m + (1-x_m)\log_2(1-x_m)\right]. \end{array}\right. \tag{8}$$

This function has a unique maximum (equal by M , for $x_m = 1/2$, $\forall m \in \overline{1,M}$) and 2^M null minims (in apexes of hyper-cube $[0,1]^M$). For example, if $M = 2$, the surface depicted in below is generated. In general, S generates a hyper-surface inside the Euclidean space $\Re^M$, but all its minima are null.

If the argument of this function is a probability distribution, it is referred to as *Shannon entropy*. If the argument is a membership function defining a fuzzy set, it is refereed to as *(Shannon) fuzzy entropy*. Denote the fuzzy entropy by S_μ. Then, according to equation (8), S_μ is expressed for all $k \in \overline{1,K}$ by:

$$S_\mu(\mathsf{R}_k) = -\sum_{i=1}^{N}\sum_{j=1}^{N} \mathsf{M}_k[i,j]\log_2 \mathsf{M}_k[i,j] - \sum_{i=1}^{N}\sum_{j=1}^{N}[1-\mathsf{M}_k[i,j]]\log_2[1-\mathsf{M}_k[i,j]]. \quad (9)$$

Obviously, this function also has a unique maximum and all minima null, with respect to variables $\mathsf{M}_k[i,j]$, its dimension being $M = N^2$.

Two main reasons motivate this choice. First, S_μ helps us make a direct connection between “how fuzzy” is a set and “how much uncertainty” it contains. Thus, since S_μ computes the quantity of information of an informational entity, say a fuzzy set, as the estimated uncertainty that the entity contains, the *minimally fuzzy sets* will subsequently contain the *minimally uncertain* information[1]. Secondly, the “total ignorance” (or uncertain) information is expressed by the unique maximum of S_μ, whereas multiple minimum points (actually, the apexes of the hyper-cube) belong to a “perfect knowledge zone” (as less uncertain information as possible). Between “total ignorance” (which, interestingly, is unique) and “perfect knowledge zone” (which is always multiple) there are many intermediate points associated to different degrees of uncertainty in knowledge about the entity.

Moreover, a *force driving towards knowledge* can be determined [10], by computing the gradient of Shannon fuzzy entropy. It is interesting to remark that the amplitude of this force (its norm), expressed as:

$$\left\|\nabla S_\mu(\mathsf{R}_k)\right\| = \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N}\left[\log_2 \frac{1-\mathsf{M}_k[i,j]}{\mathsf{M}_k[i,j]}\right]^2}\,, \quad (10)$$

increases very rapidly in the vicinity of any “perfect knowledge” point (see Fig. 6(b) above).

E. The Uncertainty Measure

Although a unique maximum of Shannon fuzzy entropy (9) exists, as proven by (10), we are searching for one of its minima. The required measure of uncertainty, V, is obtained by composing S_μ in (9) with T in (7), that is: $V = S_\mu \circ T$. Notice that V is not a measure of fuzziness, because its definition domain is the set of source-plans (crisp sets) and not the set of fuzzy relations between agents (fuzzy sets). But, since T is a bijection, the optimization problem (1) is equivalent with:

$$\mathsf{P}_{k_0} = T^{-1}(\arg\min_{k\in\overline{1,K}} S_\mu(\mathsf{P}_k))\ ,\ \textit{where}\ k_o \in \overline{1,K}\,. \quad (11)$$

The new problem (11) does not require a special optimization algorithm, since K is a finite number and all minima, although multiple, are null and localized in apexes of hyper-cube $[0,1]^{N^2}$. Problems could appear only if K is very large. In this case,

[1] Notice, however, that only the vagueness facet of the uncertainty is measured here. Ambiguity requires more sophisticated measures [7].

genetic algorithms [5] or *annealing algorithms* [6] can be used to find the minimum. According to the previous interpretations, P_{k_0} is the least fuzzy (minimally fuzzy), i.e. the least uncertain source-plan from the family and the most attracted by the knowledge zone. Its corresponding optimum fuzzy relation R_{k_0} might be useful in the construction of a *least uncertain plan* of MAS.

F. Emergence of Holonic Clusters

Once one pair ($\mathsf{P}_{k_0}, \mathsf{R}_{k_0}$) has been selected by solving the problem (11) (multiple choices could be possible, since multiple minima are available), a corresponding source-plan should be identified. Two choices are possible:

- List all the configurations of P_{k_0} (by extracting, eventually, those configurations for which the occurrence degree vanished in R_{k_0}):

$\mathsf{P}_{k_0} = \{P_{k_0,1}, P_{k_0,2}, \ldots, P_{k_0,M_{k_0}}\}$.

- Construct other source-plans by using not P_{k_0}, but R_{k_0}.

The α-cuts of R_{k_0} are the crisp relations $R_{k_0,\alpha}$, for degrees of membership $\alpha \in [0,1]$. The characteristic matrix elements of $R_{k_0,\alpha}$ are defined by:

$$H_{k_0,\alpha}[i,j] \overset{def}{=} \begin{cases} 1 \ , \text{ if } M_{k_0}[i,j] \geq \alpha \\ 0 \ , \text{ otherwise} \end{cases}, \ \forall i,j \in \overline{1,N}. \tag{12}$$

Each matrix $H_{k_0,\alpha}$ in (12) generates a unique clustering configuration of agents over A_N. Thus, two categories of source-plans emerge: *equivalence* or *holonic source-plans* (when R_{k_0} is a similarity relation) and *compatibility source-plans* (when R_{k_0} is only a proximity relation).

- When the associated fuzzy relation R_{k_0} is a *similarity* one, then an interesting property of the MAS is revealed: clusters are associated in order to form new clusters, as in a "clusters within clusters" holonic-like paradigm.

3.2 Evolution

In the open environment created by the dynamic Web opportunities for improvement of an existing virtual organization arise continuously. New partners and customers alike come into the virtual game bidding their capabilities and money to get the best deal. Staying competitive in this high dynamics requires openness and ability to accommodate chance rapidly through a flexible strategy enabling re-configuration of the organization to be able to respond to new market demands as well as to opportunities (e.g. in playing with a better partner when needed.) In response to this need we have designed an evolutionary search strategy that enables the virtual organization to continuously find better partners fitting the dynamics of its goals as they change according to the market dynamics.

A. Selection Pressure in Cyberspace

We regard ‘the living Web’ as a genetic evolutionary system. Our construction is based on the observation that the search process on an *agent domain* containing information about a set of agents that ‘live’ in the Web is analogous to the genetic selection of the most suitable ones in a population of agents meant to ‘fit’ the virtual organization goals. The mutation and crossover operators (p_m and $p_{c)}$ represent probabilities of finding ‘keywords’ (describing the attributes required from the new partners searched for) inside the search domain considered.

The main idea is to express the fitness function (measuring how well the new agent fits the holarchy’s goal) in terms of the fuzzy entropy (9):

$$\mathrm{F} = S_{\mu} \tag{13}$$

With this, minimizing the entropy across the extended MAS (which includes the agents from the search domain) according to HE goal-reach optimization equates optimizing the fitness function which naturally selects the best agents fitting the optimal organizational structure of the HE. In the sequel we present the mathematical formalism for this evolutionary search.

4 Applications of AIIs

4.1 AIIs for Global Manufacturing

Our work with the Holonic Manufacturing Systems (HMS) consortium demonstrated that this methodology is very useful for global supply chain management systems that integrate collaborative workflow techniques [18]. Within this context AIIs can be viewed as information ecosystems composed of collaborative but autonomous holons Fig. 4 working e.g. to create a new product by merging several specialized companies and coordinating their efforts, Fig. 5 (from [18]).

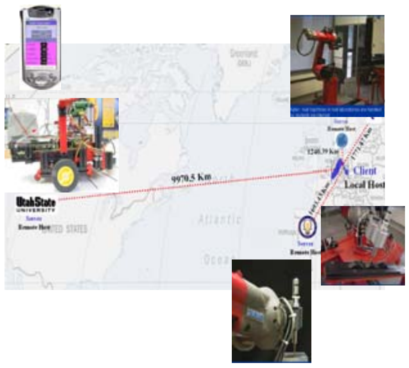


Fig. 4. Global Manufacturing Holarchy

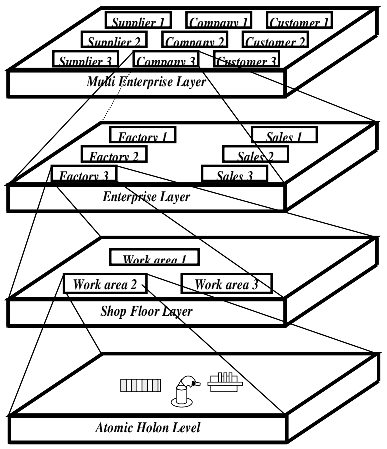


Fig. 5. Layers in Holonic Manufacturing

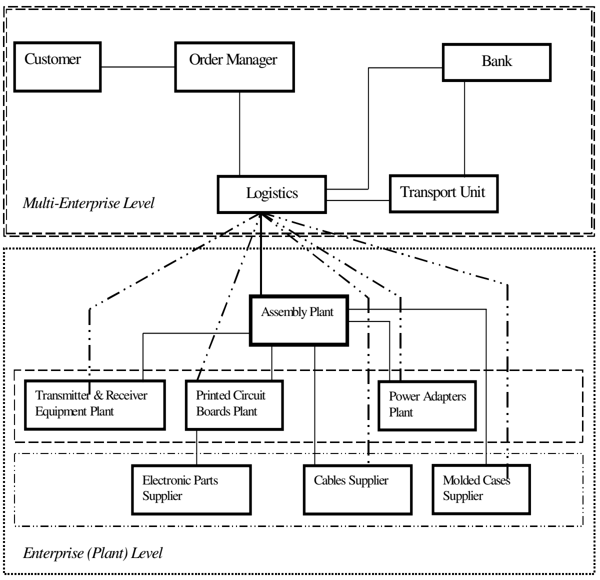


Fig. 6. The Supply Chain Holarchy

The interaction between distributed enterprises, with their suppliers and customers is modeled at the multi-enterprise level. The enterprise level hosts co-operation between entities belonging to one organization, the sales offices and the production sites. The distributed manufacturing control within a production site or shop floor is handled by the shop floor level. Here the entities are distributed work areas working together and in co-operation, in order to fulfill all orders allocated to them. The basic level (the Cell) models the interactions between equipments and humans. In [18] we focused on a supply chain scenario from the phone manufacturing industry. This approach can easily be expanded to any goods distribution networks (e.g. the Wal-Mart supply chain). Figure 6 presents the overall holarchy integrating both inter and intra- enterprise levels.

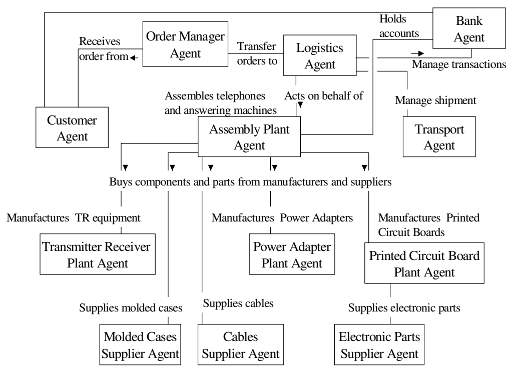

Receives order from
Order Manager Agent
Transfer orders to
Logistics Agent
Holds accounts
Bank Agent
Manage transactions
Customer Agent
Assembles telephones and answering machines
Acts on behalf of
Manage shipment
Assembly Plant Agent
Transport Agent
Buys components and parts from manufacturers and suppliers
Manufactures TR equipment
Manufactures Power Adapters
Manufactures Printed Circuit Boards
Transmitter Receiver Plant Agent
Power Adapter Plant Agent
Printed Circuit Board Plant Agent
Supplies molded cases
Supplies cables
Supplies electronic parts
Molded Cases Supplier Agent
Cables Supplier Agent
Electronic Parts Supplier Agent

Fig. 7. Conceptual Model of Supply Chain Agents

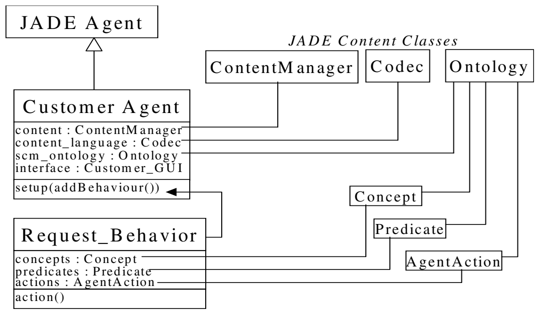


Fig. 8. Class Structure of Customer Agent

Having defined the entities involved in the overall holarchy and established the roles and their interactions within the supply chain application, we can create a network of agents (Fig. 7) based on the responsibilities that come from these roles and the resources that need to be produced or consumed.

Figure 8 shows the agent class structure of the Customer agent class, that extends the core agent of the JADE platform (www.fipa.org) thus inheriting all the functionalities that it needs to setup, register, shut down, communicate, and so on.

More details about the ontology (Fig. 9) and our system's implementation can be found in [18].

4.2 AIIs for Emergency Response Management

More recently, we successfully took the holonic concept out of the factory environment by designing a holonic framework suitable for emergency response applications [19]. For this testbed (Fig. 10 – from [28]) the actors are either a policeman with a PDA, a firefighter with a cell phone or even a helicopter sending real-time information about the traffic jams to our planner holon. For example, it can indicate an optimal or improved route for emergency vehicles to follow or even more, it will be able to instruct the policemen to clear a road so the firefighters will be able to arrive to the building faster. In case of a bigger disaster our system will be able to contact the hospitals in the zone and start distributing the patients according to bed availability. The emergency AII is depicted in Fig. 11.

During an AII-enabled rescue operation (Fig. 10), novel e-Health technologies can be used, e.g. for patient are authentication by a wireless fingerprint sensor that accesses their profile from a remote database [36]. Depending on indicators such as blood pressure and the health history of the patient, a first diagnosis will be compiled using automated decision support systems [27]. Electronic logistics support will provide information about the next available and suitable hospital, initiate staff assembly and emergency room preparation, and provide on-the-fly patient check-in. Planning and scheduling of resources on all levels of the emergency holarchy (Fig. 11) will enable reconfiguration and flexibility by selecting functional units, assigning their locations, and defining their interconnections (e.g. reallocating hospital beds to cope with the victims, rerouting around a fire crew or changing the assignments of a multi-functional defense unit).Once a crisis arises an AII emerges clustering available resources (modeled as software agents) to deal with the situation optimally [29].

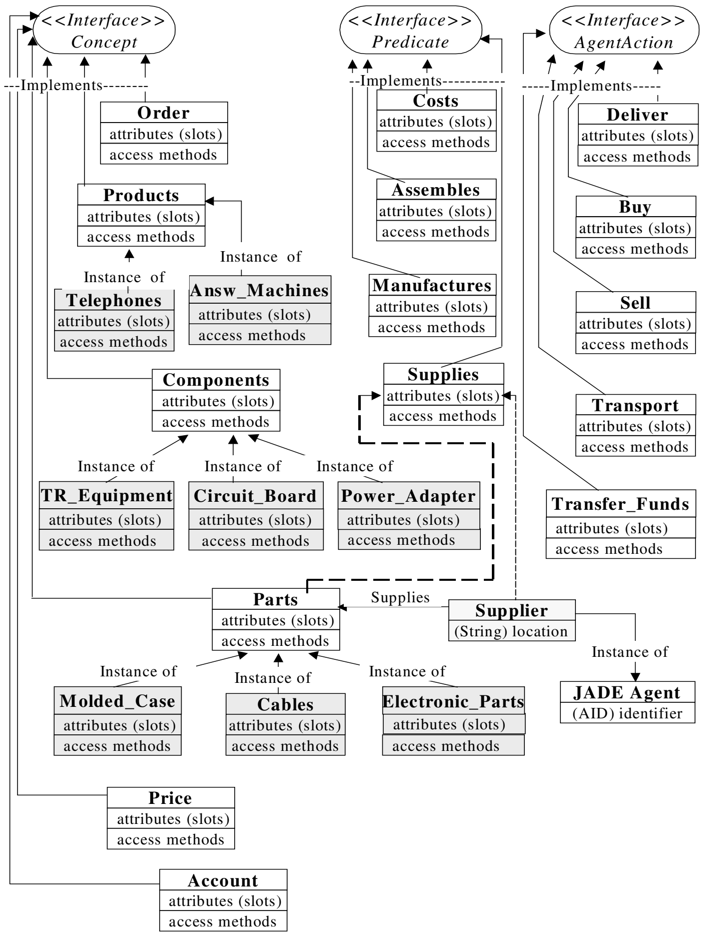


Fig. 9. Dependency relationships in the supply chain domain ontology

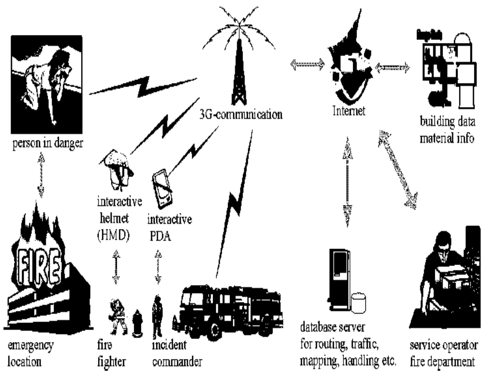


Fig. 10. Fire Emergency Scenario

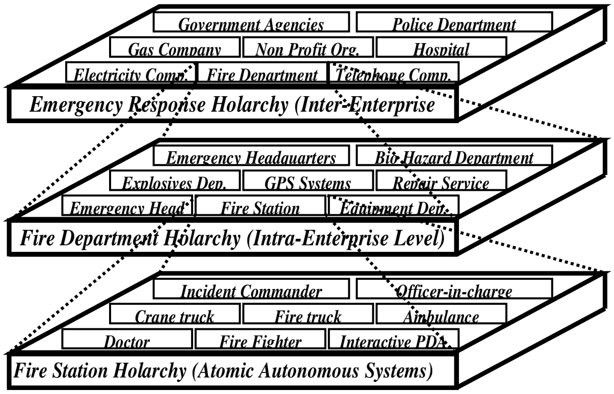


Fig. 11. Emergency Response Holarchy (AII)

4.3 Scalable Secure Web Based Services for e-Health

We propose a holonic framework suitable for e-health applications. In [20] we defined the concept of medical holarchy as an open evolutionary health system that is highly self-organized and self-adaptive. The collaborative medical entities (patients, physicians, medical devices, etc.) that work together to provide a needed medical service for the benefit of the patient form a medical holarchy [26]. The levels of a medical AII are (Fig. 12):

- ***Inter-Enterprise***: Hospitals, Pharmacies, Medical Clinics/Laboratories
- ***Intra-Enterprise***: Sections/Units/ Departments of each medical enterprise
- ***Resource Level***: Machines for medical tests, medical monitoring devices, information processing resources

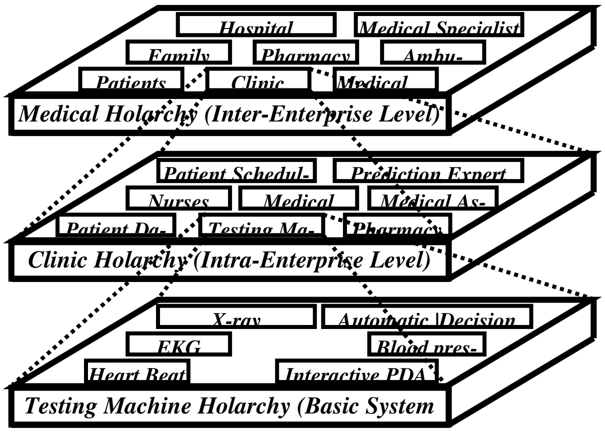


Fig. 12. Medical Holarchy

In this system of collaborative medical entities new devices and services (Fig. 12) can integrate themselves, offer their functionality to others and share data on a secure level. The complex interaction of diagnosis, treatment and monitoring is made possible through task planners and schedulers that are distributed, automatic and self-configuring.

A major issue in e-Health technology adoption is reconciliation of the various standards of care across the continents. As well the security and privacy of electronic medical records is of major significance and has proven to be the major brake that slowed down the adoption of e-Health by major clinics around the world but especially in the North Americas. Therefore our goal is to develop a reusable framework for secure high-performance web-services in e-health. As a testbed for the secure AII to be developed we will use it to connect a network of medical experts that will collaborate via the AII to develop standards of care for glaucoma [24]. To enable the collaboration of highly specialized glaucoma surgeons located across the country we have developed a telehealth approach [21] that involves a consensus analyzer synthesizing expert opinions into standards of care [22].

Recently we successfully applied this concept to improve glaucoma monitoring [23] with a security layer. This has encouraged us to expand the holonic concept to other e-Health areas that require the dynamic creation of organizational structures and workflow coordination, such as rescuing people after an accident or disaster. This is a time critical operation that requires quick diagnosis, identification of the closest available hospital and knowledge of traffic conditions.

4.4 Holonic Cybersecurity System

Information infrastructures are critical to the functioning of society; however, they are vulnerable because of threats and complex interdependencies [31]. New research in this field needs to account for these security issues, which are crucial to future information systems and services. In this context, AIIs provide new dimensions to security:

- *Reliability* of critical infrastructure with survival capabilities, such as power and water distribution.
- *Resilience* based on an anticipative environment that enables operation under continuous threats and attacks.

The issue of Cybersecurity is very difficult to tackle, given that nobody owns the Internet and there is no single ‘command post’ to control its security. The status quo regarding intrusion detection raises many challenges:

- Post attack information accumulates through many different organizations; therefore ID tools are unable to interact, making correlation of results difficult.
- Incident responses are local. There is no unified mechanism for analyzing such informational alerts and determine their implications/risk factor.

This places on the ‘wish list’ for security systems the following demands:

- ‘On-the-fly’ system configuration, requested by the continuous network changes
- Timely detection of *all kinds* of attacks
- Prevention (and counter-attack) in *any network place*
- Universal installation and maintenance

To cope with these needs, we propose a holonic cybersecurity model that emulates biological behavior by inducing immunity into the network or system under attack. Much like Noria et.al. realize network immunity in [37], our system is organized as a holarchy distributed throughout the network, Fig. 13. The AII will anticipate attacks by activating specialized agents seeking the presence of intruders into the network, similar to how antibodies fight viruses in biological systems.

- Inter-Enterprise Level

At the highest level, the "Command Post" embeds the generic security policy for an organization, which takes care of the following tasks:

- Crisis Management
- Coordinating with other organizations/government agencies
- Lower level systems management
- Shared information with trusted organizations
- Specifies which sets of network parameters should be analyzed by each entity in the holarchy3

In case of an unexpected attack, every command post in the security holarchy is alerted, triggering fighter agents that specialize in eliminating attackers.

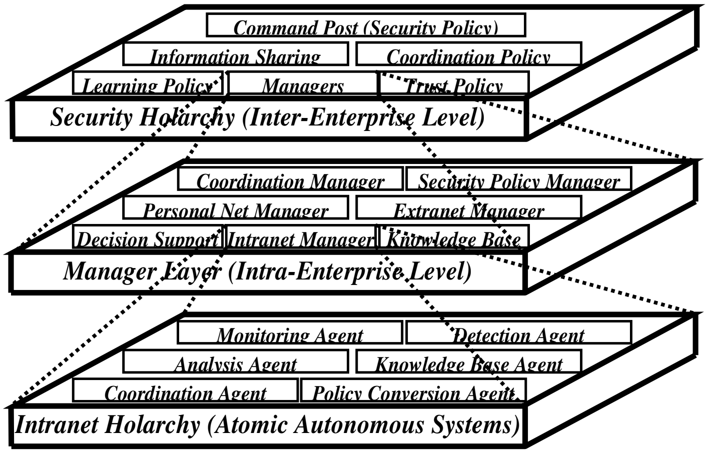


Fig. 13 Cybersecurity Holarchy

- Intra-enterprise Level

At this level managers control specified agents to analyze and correlate data collected by them, whereas at the lowest level, local agents monitor specified activities. Their main functions are:

- Understand network topology
- Analyze information given by Agents
- Make decisions depending on network topology and information given by other managers and their agents

- Coordinate the ‘atomic’ agents (e.g. scheduling their operations)
- Manage the ‘atomic agents’ knowledge base updates and mediate information exchange with the ‘command post’ (Fig. 13).

Manager agents interact with the ‘atomic’ agents by (Fig. 13):

- sending goals, derived from security policies;
- delegating specific functions of monitoring/detection and specifying the various domains to monitor;
- gathering particular information, such as: the suspicion level of a particular user, the list of events generated by a user, etc.;
- gathering relevant reports or analyses, and alarms.

- Atomic Agents Level
The basic agents have the ‘mission’ to determine an initial attack by analyzing low-level network events (‘local sniffers’). For this they carry on the following functions:

- Real-time monitoring of network packets;
- Full IP de-fragmentation and upper protocol data reassembly;
- Provide immediate information analysis in original environment, at that very instant and catching additional local data that might be required;
- Delay / block network traffic/ isolate segment suspected of ‘attack’
- Content inspection for security behavior violations
- Delete, modify suspicious/malicious content

The holonic approach enables also a topology-oriented approach in which critical points of action are identified where agents ‘migrate’ as needed. This enables in addition to the automatic detection of an attack, also attack localization as close as possible to its source. Agents must be able to isolate a specific network’s segments. Managers coordinate the activity of basic agents by moving the basic agents across different network points in order to investigate what is the really “relevant” information and how to extract quality from quantity. For implementation details see [30].

5 Conclusions

We propose a theoretical foundation for the design of adaptive information infrastructures (AIIs) enabling and sustaining tomorrow’s e-Society, as well as envision various areas of industrial application for such AIIs, that would improve human life. The recent theoretical results obtained by us in modeling the property of emergence in self-organizing systems were refined and expanded with other recent results to create a model of *emergence in Cyberspace*, by this setting a foundation for engineering self-organizing applications mirroring biological behavior.

The principal merit of the proposed holonic AII architecture is that it provides an environment that can react appropriately to highly unpredictable situations. By using natural models of emergence, much in the same manner as DNA is controlled in genetic engineering, we will be able to control the emergence of AIIs as crises arise. AIIs will address the emergency quickly, efficiently and most appropriately. Once a goal is set (where a certain need has to be fulfilled), the AII self-organizes to accomplish this goal optimally.

Some of the difficult questions posed by this research are:

- Can pathological emergent behavior of the total system, arising from the interactions between people, agents, objects, and their various policies, be avoided?
- How do we translate the interaction of agents in different contexts and environments into machine understandable language?
- How do we express and code sufficient real world semantics when the scope of interaction between agents is too broad or not predefined [35]?

References

[1] Stuart Kaufmann, At Home in The Universe: The search for the laws of self organization and complexity, New York: Oxford University Press, 1995

[2] Werner E (1996); 'Logical Foundations of Distributed Artificial Intelligence' in Foundations of Distributed Artificial Intelligence (eds. O'Hare G.M.P. and Jennings N.R.) (1996) John Wiley & Sons Interscience.

[3] Zimmermann, H-J, – *Fuzzy Set Theory And Its Applications*, Kluwer Academic (1991).

[4] Klir, G. and Folger, Tina, *Fuzzy sets, Uncertainty, and Information*, Prentice Hall, (1988).

[5] Goldberg, D.E., Simple Genetic Algorithms, University of Michigan, Dept. of Civil Engineering, Ann Arbor, MI (1982)

[6] S. Kirkpatric, C.D. Gelatt, M.P. Vecchi, Optimization by Simulating Annealing, Science, 220:671-680 (1983)

[7] Stuart Kaufmann, *Investigations*, Oxford University Press, ISBN 0-19-512104-X

[8] Mihaela Ulieru, "Emergence of Holonic Enterprises from Multi-Agent Systems: A Fuzzy-Evolutionary Approach", Invited Chapter in *Soft Computing Agents: A New Perspective on Dynamic Information Systems*, (V. Loia – Editor), IOS Press -Frontiers in AI and Applications Series 2002, ISBN 1 58603 292 5, pp. 187-215.

[9] Mihaela Ulieru, "Modeling Holarchies as Multi-Agent Systems to Enable Global Collaboration", Proceedings of the IEEE Computer Society Press – 13th International Conference and Workshop on Database and Expert Systems Applications (DEXA 2002), September 2-6, 2002, Aix-en-Provence, France, pp. 603-608, ISBN 0-7695-1668-8, Order # PRO1668.

[10] Mihaela Ulieru, Dan Stefanoiu and Douglas Norrie, "Holonic Metamorphic Architectures for Manufacturing: Identifying Holonic Structures in Multi-Agent Systems by Fuzzy Modeling", Invited Chapter in *Handbook of Computational Intelligence in Design and Manufacturing* (Jun Wang & Andrew Kussiak – Editors), CRC Press 2000, ISBN No 0-8493-0592-6, pp. 3-1 – 3-36.

[11] Mihaela Ulieru and Dan Stefanoiu, "Holonic Self-Organization of Multi-Agent Systems by Fuzzy Modeling with Application to Intelligent Manufacturing", IEEE-SMC 2000, Nashville, USA, October, pp. pp. 1661-1666 – with Dan Stefanoiu – postdoctoral fellow and Douglas Norrie.

[12] Arthur Koestler, *The Ghost in the Machine*, MacMillan, 1968.

[13] Christensen, James H., Holonic Manufacturing Systems: Initial Architecture and Standards Directions, in *Proceedings of the First European conference on Holonic Manufacturing systems*, European HMS Consortium, Hanover, Germany, 1994.

[14] Mihaela Ulieru, Scott Walker and Robert Brennan, "Holonic Enterprise as a Collaborative Information Ecosystem", Workshop on "Holons: Autonomous and Cooperative Agents for the Industry", Autonomous Agents 2001, Montreal, May 29, 2001, pp. 1-13.

[15] Mihaela Ulieru, "A Fuzzy Mathematics Approach to Modeling Emergent Holonic Structures", Invited Chapter in *Geometry, Continua and Microstructures*, pp. 241-255, Academic Press, 2002 – ISBN 973-27-0880-8.

[16] Mihaela Ulieru and Silviu Ionita, "Soft Computing Techniques for the Holonic Enterprise", FLINT 2001, M. Nikravesh and B. Azvine (Eds.), New Directions in Enhancing the Power of the Internet, UC Berkeley Electronics Research Laboratory, Memorandum No. UCB/ERL M01/28, August 2001. pp 182-187.

[17] LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace, K. Sycara, S. Widoff, M. Klusch and J. Lu, *Autonomous Agents and Multi-Agent Systems*, Volume 5, No. 2, June 2002, Kluwer ISSN 1387-2532.

[18] Mihaela Ulieru and Mircea Cobzaru, "Building Holonic Supply Chain Management Systems: An e-Logistics Application for the Telephone Manufacturing Industry", IEEE Transactions on Industrial Electronics, December 2004 (accepted).

[19] Mihaela Ulieru and Rainer Unland, "Emergent e-Logistics Infrastructure for Timely Emergency Response Management by Collaborative Problem-Solving with Optimized Resource (Re)Allocation", invited chapter in *Engineering Self-Organising Systems - Nature-Inspired Approaches to Software Engineering*, Di Marzo Serugendo, G.; Karageorgos, A.; Rana, O.F.; Zambonelli, F. (Editors), Springer Verlag, 2004, X, 299 ISBN: 3-540-21201-9, pp. 139-156.

[20] Mihaela Ulieru, "Internet-Enabled Soft Computing Holarchies for e-Health Applications", in New Directions in Enhancing the Power of the Internet, (L.A. Zadeh and M. Nikravesh – Editors), pp. 131-166, Springer Verlag, Berlin, 2003.

[21] Mihaela Ulieru and Alexander Grabelkovsky, "Telehealth Approach to Glaucoma Progression Monitoring", *International Journal of Information Theories and Applications* 10(3), 2003, ISSN 1310-0513, pp. 326-330.

[22] Mihaela Ulieru and Marcelo Rizzi A Cooperative Approach to the Development of Expert Knowledge Bases Applied to Define Standard of Care in Glaucoma, Proceedings of CoopIS 2003, Catania, Sicily, Nov. 3-7, 2003, pp. 235-243, Springer Verlag Lecture Notes in Computer Science LNCS 2888.

[23] Mihaela Ulieru, Soft Computing Agents for e-Health, NAFIPS 2004 (North-American Fuzzy Information Processing Society) International Conference, Banff, Canada, June 27-30 (accepted).

[24] Mihaela Ulieru and Adam Geras, "Emergent Holarchies for e-Health Applications – A Case in Glaucoma Diagnosis", Proceedings of IECON 2002 – 28th Annual Conference of the IEEE Industrial Electronics Society, November 5-8, 2002, Seville, Spain, ISBN 0-7803-7475-4, pp. 2957-2962, (proceedings on CD-Rom, IEEE Catalog Number 02CH37363.)

[25] Maja Hajdec, Elizabeth Chang and Mihaela Ulieru, Ontology-Based Holonic Medical Diagnostic System, IASTED International Conference on Biomedical Engineering, Innsbruk, Austria, February 15-18, 2005 (submitted).

[26] Mora, T. and Ulieru, M., Agent-Based Decision Support Systems for the Industry, Proceedings of INDIN 2004, Second International Conference on Industrial Informatics, Berlin, Germany, June 24-26, 2004, pp. 391-396.

[27] Mentrup, C. and Fuhrer, O., e-Motion: e-Mobile Testbed for Interoperability of Networks in e-Logistics, Proceedings of the 1st International Conference on Mobile Business, July 8-9, 2002 Athens, Greece.

[28] Mihaela Ulieru and Paul Worthington, Adaptive Risk Management System for Critical Infrastructure Protection, International Journal of Integrated Computer Systems Engineering, Special Issue on Autonomic Computing (accepted) 2005.

[29] Mihaela Ulieru, Emergent Computing for the Industry: Agents, Self-Organization and Holonic Systems, Proceedings of IEEE-IECON 2004, November, Busan, Corea.

[30] Tom Berson, Sun Tzu in Cyberspace: The Art of Information Warfare, Keynote Address at the Cybersecurity 2003 Conference, May 20, Foster City, CA, USA.

[31] Tim Kindberg and Armando Fox, Systems Software for Ubiquitous Computing, *IEEE on Pervasive Computing*, Jan-Mar 2002, pp. 70-81.

[32] http://www.biometrics.org

[33] Noria Foukia and Salima Hassas and Serge Fenet and Paul Albuquerque, "Combining Immune Systems and Social Insect Metaphors: A Paradigm for Distributed Intrusion Detection and Response System" in Proceedings of Mobile Agents for Telecommunication Applications, 5th International Workshop, MATA 2003, Marakech, Morocco, October 8-10, 2003

Agent-Based Modelling of Stem Cell Self-organisation in a Niche

Mark d'Inverno and Rob Saunders

Cavendish School of Computer Science,
University of Westminster, 115 New Cavendish Street,
London W1M 8JS
{dinverm, saunders}@wmin.ac.uk

Abstract. It is our belief that modelling the behaviour of stem cells in the adult human body as an agent-based system is the most appropriate way of understanding the process of self-organisation. We have undertaken several case studies where formal and/or computational models of stem cell systems, have been re-developed using an agent-based approach. This paper presents details of one of these case studies where we have used an agent-based approach as opposed to a cellular automata approach. A formalisation of the non-agent and agent-based approach is given, and from the results of this investigation, we aim to demonstrate the advantages of the agent-based approach for developing biologically plausible models with emergent self-organising dynamics. The aim of this paper first to discuss the importance of modelling and simulating stem cells, because of certain experimental limitations, but also to demonstrate that the multi-agent approach to modelling is the most appropriate.

1 Introduction

In recent years there has been a growing debate about how stem cells behave in the human body; whether the fate of stem cells is pre-determined [11] or stochastic [13, 19], and whether the fate of cells relies on their internal state [12], or on extra-cellular micro-environmental factors [21]. There have been several attempts to build formal models of these theories, so that predictions can be made about how and why stem cells behave either individually or collectively. An excellent review of these formal approaches can be found in a recent publication [22].

Recent experimental evidence has suggested that stem cells development may be more complicated than was originally thought. The standard model of stem cell development is that a stem cell becomes increasingly differentiated over time along a well-defined cell lineage and eventually becomes a fully functional cell. This model has been challenged by many researchers including one of our collaborators, Neil Theise [7, 18, 16]. Several years ago, new theories were proposed by our collaborator and others that challenged the prevailing view because new experimental data suggested that stem cell fate is both *reversible*, i.e. cells can become less differentiated or behave more like stem cells, and *plastic*, i.e. cells can migrate from one cell lineage to another.

Whilst working on Cell, with an interdisciplinary team including Theise and the artist Jane Prophet, it became clear to us that the most appropriate way to model stem cells in the adult human body was as a dynamic system of self-organising agents. Our work to date has used our existing, well-established techniques for specifying and modelling agent-based systems in general [4, 6, 9, 10] and progressed along two parallel strands. The first strand of our work has been an attempt to develop an agent-based model of Theise's theory of stem cell behaviour and organisation [17, 5]. The second strand has been to use the same agent-based approach to analyse and re-develop existing models to ensure that our framework is sufficiently flexible to model more than one theory and to understand how other work differs from our own. In other words, we have been working on re-implementing agent-based versions of cellular automata and equational models of stem cells in order to support our claim that the agent approach is more suitable than other current modelling approaches.

In this paper we consider one of the latest models of stem cell systems and show what can be gained from evaluating them using our agent framework. The aim of this paper is to show why we need simulations of stem cell behaviour in general, to demonstrate the role of formal modelling in developing these simulations, and to show the benefits of a multi-agent approach over other possible modelling approaches. We also aim to substantiate our belief that stem cell self-organisation and behaviour is an emergent of the individual interactions of individual stem cells with each other and with the environment in which they are situated.

Before we consider this work in detail, we first consider the reasons why we might want to build models of stem cell systems in general.

1.1 Formal Modelling of Stem Cells

The mathematical modelling, conceptualisation and simulation of stem cell behaviour is beginning to receive a substantial amount of interest from a number of researchers [14, 1, 8]. As has been pointed out by others, predictive models of stem cell systems, could provide important new understandings of the self-regulating mechanisms that result in well known global properties of stem cells. These include the following qualities of a healthy human adult.

1. There are always a sufficient number of stem cells.
2. Fully determined cells are sufficiently replenished as they die.
3. The system of stem cells can recover after serious injury or disease.

As has been discussed by a number of authors [22, 14] there are several reasons why formal predictive models of stem cells will receive an increasing amount of attention in the near future. Though the first model we know of was published in 1964 ([20]) there has been surprisingly little work in this field until the last couple of years. Indeed over the last few years, there has been a noticeable climate change in this respect, and there is now a growing awareness of the need to use mathematical modelling and computer simulation to understand the processes and behaviours of stem cells in the body. An excellent review of existing models has been recently published [22].

We summarize what we see are the key reasons for the systematic development of formal models and simulations to consider hypothesis about the nature and behaviour of stem cells.

1. It is not possible to investigate how stem cells react by looking at dead tissue, and much stem cell research is based on observation of dead, 2-D slides. Building simulations allows researchers to test possible cell behaviours that can then be related back to observable laboratory results.
2. In the adult body, stem cells cannot be distinguished morphologically from other primitive non-determined cell types. It is therefore hard, if not impossible, to observe their behaviour in the dynamic system of which they are a part.
3. The size and complexity of stem cell systems mean that without simulation, it is not possible to consider the whole system. Simulations provide an important tool for understanding the global behaviour of complex systems reacting agents.
4. Clearly any formal model, and resulting simulation, of stem cells will necessarily incur massive simplifications and abstractions about the machinations of the human body. It is our belief, however, that theoretical simplifications are often key to understanding fundamental properties of natural systems.
5. It is the *potential* of cells to behave in lots of different ways which makes them more or less stem like. It may be that stem cell is a notion rather than an artifact and refers to the wide-ranging set of potential behaviours that it might have that are influenced by internal, environmental, and stochastic processes. Simulations provide a way of determining which behaviours are essential to stem cells and which are incidental in systems that have been studied in the laboratory.
6. When you consider experimental evidence you have seen only one behaviour. This behaviour may have been one of many, and it is the potential for cells to behave in certain ways that might be key to defining them. Modelling and simulation is a much more effective device for understanding "behavioural potential" than looking at completed chains of events in the lab.
7. Though our work has been explicitly concerned with modelling the adult human body, it is clear that simulation does not involve any ethical difficulties such as extracting stem cells from an embryo in such a way that it is sacrificed.
8. And of course, simulation is cheap.

This should give the reader an indication of why we believe this will become a growing field in the next few years. In our approach we have used an agent-based approach to the formal modelling and simulation of stem cells, and we make the following claims which we will attempt to substantiate in this paper.

1. An agent-based approach provides more flexibility than other more limited approaches and so delivers greater potential for modelling more sophisticated, globally emergent, behaviour.
2. An agent-based approach can also provide more biological plausibility than existing approaches such as cellular automata and other mathematical approaches. One of the main reasons that biological plausibility is important is to attract biologists to use and work with any models and simulations that are created.
3. Stem cells are a prime example of a self-organising system where individual cells react to their local physical, chemical and biological environment. The system should therefore be most suitably modelled as a system of interacting reactive

agents, where the reaction at the micro level gives rise to the emergent behaviour at the system level.

4. Even though we are simulating cells and environment, the Brooksian idea of an agent being something which is both situated and embodied ([2]), is a fundamental driving force of our use of agents as the appropriate modelling paradigm. Cells modelled as agents have a physical, chemical and biological presence and are situated in a physical, chemical and biological environment in which they react. The way in which they react will then influence the way other cells react in the future and so on. This then, becomes a complex system, as we have claimed before, that stem cell systems should be modelled as complex adaptive systems.
5. By situating our simulation work in a wider formal framework we can compare and evaluate different models. We believe that this is necessary for this new field to develop in a systematic manner.
6. Moreover, the formal framework allows us to "agentify" existing models, making it very clear what the relationship between the existing version and the agent version is.
7. By building a formal model using a specification language from software engineering, there are techniques to ensure that the simulation correctly implements the model.

In this paper we go some way to justifying our claims above by looking at one case study in detail. We consider the work of Agur et al. who have developed a cellular automata model of stem cells, and show that by re-caging this work in terms of an agent model, we can highlight difficulties of the cellular automata approach in general, but also increase the biological plausibility of the model.

In what follows below we will provide formal specifications of the original model and the agent-based reformulation using the language Z [15]. We have a history of using Z to build specifications of agent and non-agent computational systems that allows us to compare and evaluate different models and approaches [6].

2 A Cellular Automata Approach to Modelling Stem Cells

In recent work, Agur et al. [1] built a cellular automata model to show how the number of stem cells in the bone marrow could be maintained and how they could produce a continuous output of determined cells. The bone marrow is considered to be a stem cell *niche* where most biologists believe that the human body's supply of hematopoietic stem cells are situated and maintained.

This work is important because it is one of the few examples where a mathematical model has been used to show what properties of stem cells might be required to enable the maintenance of the system's homeostasis. The model demonstrates a possible mechanism that allows a niche to maintain a reasonably fixed number of stem cells, produce supply of mature (determined) cells, and to be capable of returning to this state even after very large perturbations that might occur through injury or disease. The behaviour of a cell is determined (equally differentiated) by both internal (intrinsic) factors, e.g. a local counter, and external (extrinsic) factors, e.g. the prevalence of stem cells nearby, as stated by the authors as follows.

1. Cell behaviour is determined by the number of its stem cell neighbours. This assumption is aimed at simply describing the fact that cytokines, secreted by cells into the micro-environment are capable of activating quiescent stem cells into proliferation and determination.
2. Each cell has internal counters that determine stem cell proliferation and stem cell transition into determination as well as the transit time of a differentiated cell before migrating to the peripheral blood.

In the cellular automata model, the niche is modelled as a connected, locally finite, undirected graph.

$$[Node]$$

$$\begin{array}{|l}
[X] \\
\hline
connected\ \_ : \mathbb{P}(\mathbb{P}(X \times X)) \\
\hline
\end{array}$$

This can be represented as a symmetric relation on the set of nodes, such that no node relates to itself. We also assume that a graph is connected.

$$\begin{array}{|l}
graph : Node \leftrightarrow Node \\
neighbours : Node \rightarrow (\mathbb{P}\, Node) \\
\hline
\forall\, n : Node \bullet (n, n) \notin graph \\
graph^{\sim} = graph \\
\forall\, n : Node \bullet neighbours\ n = \mathrm{ran}(\{n\} \lhd graph) \\
connected\ graph
\end{array}$$

Any $Node$ is either empty, or it is occupied by either a stem cell or a determined cell. Here we introduce a naming convention that we shall use throughout where we add a two letter suffix to all names specific to a model, in the case of the Agur model we add the suffix "Ag".

$$TypeAg ::= EmptyAg \mid StemAg \mid DeterminedAg$$

The state of any node is given by the node location, the state, and an internal clock.

$$\begin{array}{|l}
NodeStateAg \\
\hline
node : Node \\
type : TypeAg \\
counter : \mathbb{N} \\
\hline
\end{array}$$

The set of all such nodes is then given below, and defines the system state. We also define a function that returns the neighbouring node states for any given node state.

$$\begin{array}{|l}
\textit{SystemStateAg} \\
\hline
\textit{nodes} : \mathbb{P}\, \textit{NodeStateAg} \\
\textit{neighboursAg} : \textit{NodeStateAg} \rightarrow \mathbb{P}\, \textit{NodeStateAg} \\
\hline
\{n : \textit{nodes} \bullet n.\textit{node}\} = \textit{Node} \\
\#\textit{nodes} = \#\textit{Node} \\
\forall\, n, m : \textit{NodeStateAg} \bullet \\
\qquad m \in (\textit{neighboursAg}\ n) \Leftrightarrow \\
\qquad m.\textit{node} \in (\textit{neighbours}\ n.\textit{node}) \\
\hline
\end{array}$$

There are three constant values, we will call them *LeaveNicheAg*, *CyclingPhaseAg* and *NeighbourEmptyAg* in our specification, that are used to reflect experimental observation. *LeaveNicheAg* represents the time taken for a determined cell to leave the niche. *CyclingPhaseAg* represents the cycling phase of a stem cell; a certain number of ticks of the counter are needed before the cell is ready to consider dividing. Finally, *NeighbourEmptyAg* represents the amount of time it takes for an empty space that is continuously neighboured by a stem cell, to be populated by a descendent from the neighbouring stem cell.

$$\mid\ \textit{LeaveNicheAg}, \textit{CyclingPhaseAg}, \textit{NeighbourEmptyAg} : \mathbb{N}$$

We now specify how the system changes over time. Whenever there is a change of state in the system, we identify the node that we are considering as *node*. As a consequence of each change *node* is removed and replaced with a new node, *newnode*, that represents the updated state. All locations are updated simultaneously.

$$\begin{array}{|l}
\Delta\textit{SystemStateAg} \\
\hline
\textit{SystemStateAg} \\
\textit{SystemStateAg}' \\
\textit{node}, \textit{newnode} : \textit{NodeStateAg} \\
\hline
\textit{nodes}' = (\textit{nodes} \setminus \{\textit{node}\}) \cup \{\textit{newnode}\} \\
\hline
\end{array}$$

The rules of this model, which determine what happens at a node based on internal and external factors are described and specified below.

1. Determined cell nodes
   (a) If the internal counter of a node representing a determined cell has reached *LeaveNicheAg* then the cell leaves the niche; the internal counter of the node is reset to 0, and the new state at the node becomes empty.

$$\begin{array}{|l}
\textit{DeterminedLeaveNicheAg} \\
\hline
\Delta\textit{SystemStateAg} \\
\hline
\textit{node}.\textit{type} = \textit{DeterminedAg} \\
\textit{node}.\textit{counter} = \textit{LeaveNicheAg} \\
\textit{newnode}.\textit{type} = \textit{EmptyAg} \\
\textit{newnode}.\textit{counter} = 0 \\
\hline
\end{array}$$

(b) If the internal counter has not yet reached *LeaveNicheAg* then the internal conter is incremented.

$$\begin{array}{|l}
\hline
\textit{DeterminedStayNicheAg} \\
\Delta \textit{SystemStateAg} \\
\hline
\textit{node.type} = \textit{DeterminedAg} \\
\textit{node.counter} < \textit{LeaveNicheAg} \\
\textit{newnode.type} = \textit{node.type} \\
\textit{newnode.counter} = \textit{node.counter} + 1 \\
\hline
\end{array}$$

2. Stem cells nodes
   (a) If the internal counter of a node representing a stem cell has reached the constant *CyclingPhaseAg*, and all of the nodes neighbours are stem cells, then the state of the node becomes a determined cell and the internal counter is reset to 0.

$$\begin{array}{|l}
\hline
\textit{StemToDeterminedNodeAg} \\
\Delta \textit{SystemStateAg} \\
\hline
\textit{node.type} = \textit{StemAg} \\
\textit{node.counter} = \textit{CyclingPhaseAg} \\
\forall\, n : (\textit{neighboursAg node}) \bullet n.\textit{type} = \textit{StemAg} \\
\textit{newnode.type} = \textit{DeterminedAg} \\
\textit{newnode.counter} = 0 \\
\hline
\end{array}$$

   (b) If the internal counter of a node representing a stem cell is equal to *Cycling PhaseAg* but not all the node's neighbours are stem cells then do nothing; leave the internal counter unchanged.

$$\begin{array}{|l}
\hline
\textit{RemainAsStem1Ag} \\
\Delta \textit{SystemStateAg} \\
\hline
\textit{node.type} = \textit{StemAg} \\
\textit{node.counter} = \textit{CyclingPhaseAg} \\
\neg\, (\forall\, n : (\textit{neighboursAg node}) \bullet n.\textit{type} = \textit{StemAg}) \\
\textit{newnode.type} = \textit{node.type} \\
\textit{newnode.counter} = \textit{node.counter} \\
\hline
\end{array}$$

   (c) If the counter has not reached *CyclingPhaseAg* then do nothing except increment counter by 1.

$$\begin{array}{|l}
\hline
\textit{RemainAsStem2Ag} \\
\Delta \textit{SystemStateAg} \\
\hline
\textit{node.type} = \textit{StemAg} \\
\textit{node.counter} < \textit{CyclingPhaseAg} \\
\textit{newnode.type} = \textit{node.type} \\
\textit{newnode.counter} = \textit{node.counter} + 1 \\
\hline
\end{array}$$

3. Empty nodes
   (a) If the internal counter at an empty node has reached *NeighbourEmptyAg* and there is a stem cell neighbour then introduce, i.e. give birth to, a stem cell in that location. The internal counter of the node is reset to 0.

$$
\begin{array}{|l}
\textit{BecomeStemAg} \\
\hline
\Delta \textit{SystemStateAg} \\
\hline
\textit{node.type} = \textit{EmptyAg} \\
\textit{node.counter} = \textit{NeighbourEmptyAg} \\
\exists\, n : (\textit{neighboursAg node}) \bullet n.\textit{type} = \textit{StemAg} \\
\textit{newnode.type} = \textit{StemAg} \\
\textit{newnode.counter} = 0 \\
\hline
\end{array}
$$

   (b) If the counter at an empty grid has not reached *NeighbourEmptyAg* and there is exists a stem cell neighbour then increment the counter by 1.

$$
\begin{array}{|l}
\textit{RemainEmpty1Ag} \\
\hline
\Delta \textit{SystemStateAg} \\
\hline
\textit{node.type} = \textit{EmptyAg} \\
\textit{node.counter} < \textit{NeighbourEmptyAg} \\
\exists\, n : (\textit{neighboursAg node}) \bullet n.\textit{type} = \textit{StemAg} \\
\textit{newnode.type} = \textit{EmptyAg} \\
\textit{newnode.counter} = \textit{node.counter} + 1 \\
\hline
\end{array}
$$

   (c) If there are no stem cell neighbours at all then reset the internal counter to 0.

$$
\begin{array}{|l}
\textit{RemainEmpty2Ag} \\
\hline
\Delta \textit{SystemStateAg} \\
\hline
\textit{node.type} = \textit{EmptyAg} \\
\neg\, (\exists\, n : (\textit{neighboursAg node}) \bullet n.\textit{type} = \textit{StemAg}) \\
\textit{newnode.type} = \textit{EmptyAg} \\
\textit{newnode.counter} = 0 \\
\hline
\end{array}
$$

2.1 Discussion About the Cellular Automata Approach

We now have provided a specification of this system, and this formal model immediately identifies a number of issues with this cellular automata work.

1. The specification clearly reveals that niche spaces, i.e. empty nodes, must have counters for this model to work. In a sense, empty space is having to do some computational work. Clearly this lacks biological feasibility and is against what the authors state about modelling cells, rather than empty locations, having counters.
2. Stem cell division is not explicitly represented, instead stem cells are brought into being by empty space.

3. More subtly, these stem cells appear when empty nodes have been surrounded by at least one stem cell for a particular period of time. However, the location of the neighbouring stem cell can vary at each step. Even though the model details the fact that if a stem cell is next to an empty space long enough then it will divide so that it's descendent occupies this space. However, the rule does not state that the neighbouring stem cell must be the same stem cell for every tick of the counter. It states something much weaker; that there must be a neighbouring cell, possibly different each time, for each tick of the counter from 1 to $NeighbourEmptyAg$. Biologically, it would seem more intuitive that the same stem cell should be next to an empty niche space for this length of time in order for "division" to occur into the space but the model lacks a "directional component".
4. The state of a stem cell after division is not defined. Let us for a moment assume that the neighbouring stem cell (S) is fixed for all counts from 1 to $NeighbourEmptyAg$ from some specific location (N). Nothing is said about what happens to S after a new stem cell appears in N. For example, should the counter of S be reset after division? Neither does it give any preconditions on S. For example, does S's local counter need to have reached an appropriate point in its cycling phase for this to happen?

So the basic problem is that this model relies on allowing both unfilled niche locations as well as stem and determined cells to have counters. Moreover, it does not investigate or model the nature of a stem cell before and after division. We now attempt to re-interpret these rules using an agent-based approach that still retains the overall qualities of the model.

2.2 Re-formulation Using an Agent-Based Approach

One of the biggest differences between the original cellular automata model and our re-formulation is the change in the role of graph nodes. In the cellular automata model each node represents either a cell or an empty space. In our re-formulation, each node represents a space that may or may not contain an agent that represents a cell. This difference in the two models is illustrated in Figure 1.

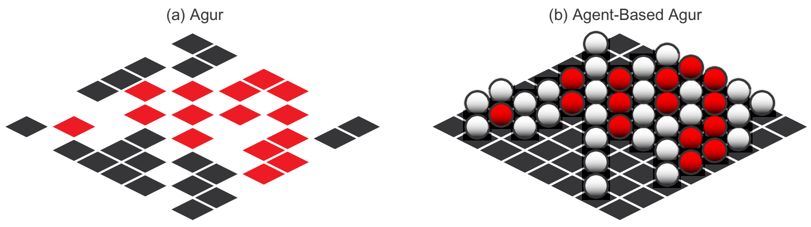


Fig. 1. A comparison of the original Agur cellular automata model and our reformulation as a grid-based agent model. In the original model the nodes maintain the state of the cells, whereas in our re-formulation the nodes contain agents and it is the agents that maintain the state of the cells

With the agent approach we also provide each cell with a unique identifier. We model all cells as having one internal counter as before. In addition there is a counter associated with each of the neighbouring nodes. The counters associated with neighbouring nodes record how long the neighbouring location has been empty. Moreover, cells can sense the type of cell at each of its neighbours, although this perception ability is only used by stem cells. If an agent represents a stem cell then it can potentially divide into any location where the counter has reached $\mathit{NeighbourEmptyAg}$.

$[\mathit{AgentId}]$

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{AgentCellAg}} \\ \hline
\mathit{id} : \mathit{AgentId} \\
\mathit{type} : \mathit{TypeAg} \\
\mathit{counter} : \mathbb{N} \\
\mathit{nscounter} : \mathit{Node} \nrightarrow \mathbb{N} \\
\mathit{nstype} : \mathit{Node} \nrightarrow \mathit{TypeAg} \\ \hline
\mathit{type} = \mathit{StemAg} \lor \mathit{type} = \mathit{DeterminedAg} \\
\operatorname{dom} \mathit{nscounter} = \operatorname{dom} \mathit{nstype} \\
\forall\, n : \mathit{Node} \mid \mathit{nstype}\ n \neq \mathit{EmptyAg} \bullet \mathit{nscounter}\ n = 0 \\ \hline
\end{array}$$

A stem cell agent is defined as follows.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{AgentStemCellAg}} \\ \hline
\mathit{AgentCellAg} \\ \hline
\mathit{type} = \mathit{StemAg} \\ \hline
\end{array}$$

A determined cell agent is defined as follows.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{AgentDeterminedCellAg}} \\ \hline
\mathit{AgentCellAg} \\ \hline
\mathit{type} = \mathit{DeterminedAg} \\ \hline
\end{array}$$

The initial state of an stem cell agent is defined as follows.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{InitAgentStemCellAg}} \\ \hline
\mathit{AgentCellAg} \\ \hline
\mathit{counter} = 0 \\
\operatorname{ran} \mathit{nscounter} = \{0\} \\ \hline
\end{array}$$

The initial state of a determined cell agent is defined as follows.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{InitAgentDeterminedCellAg}} \\ \hline
\mathit{AgentDeterminedCellAg} \\ \hline
\mathit{counter} = 0 \\ \hline
\end{array}$$

We define a mature stem cell as one which is ready to divide.

$$
\begin{array}{|l}
\textit{MatureAgentStemCellAg} \\
\hline
\textit{AgentStemCellAg} \\
\hline
\textit{counter} = \textit{CyclingPhaseAg} \\
\hline
\end{array}
$$

The system state consists of the niche where some nodes are filled with cells. The first predicate simply states that the empty nodes are those nodes which do not contain a cell. The second predicate states that the neighbours are defined by the graph to which the cells are attached.

$$
\begin{array}{|l}
\textit{AgentSystemStateAg} \\
\hline
\textit{cells} : \textit{Node} \rightarrow \textit{AgentCellAg} \\
\textit{emptynodes} : \mathbb{P}\, \textit{Node} \\
\hline
\textit{emptynodes} = \textit{Node} \setminus (\mathrm{dom}\, \textit{cells}) \\
\forall\, n : \textit{Node};\ c : \textit{AgentCellAg} \mid (n, c) \in \textit{cells} \land c.\textit{type} = \textit{StemAg} \bullet \\
\qquad \mathrm{dom}\, c.\textit{nscounter} = \mathrm{ran}(\{n\} \lhd \textit{graph}) \\
\hline
\end{array}
$$

2.3 Operation

Space does not permit us giving a full treatment, and of course many of the operations would be identical to that which we have specified before, but we outline the basic operations here.

1. Cells set/update counters.
2. Mature stem cells that are surrounded by empty neighbours and have neighbour counters have reached *NeighbourEmptyAg* will make a request to the environment to divide into two daughter stem cells.
3. The environment resolves any conflicts where several cells wish to divide into the same node and informs those mature stem cells that can divide and those that are not able to.
4. Mature stem cells that are able to divide do so. Mature stem cells that are surrounded by stem cells become new determined cells. Mature determined cells which are ready to leave the niche do so.

We consider each of these four stages in turn.

Updating Counters. We use the auxiliary function which increments all the counters of a cell up to the maximum value.

$$
\begin{array}{|l}
\textit{incrementcounters} : (\textit{Node} \mathrel{+\!\!\!\!\rightarrow} \mathbb{N}) \rightarrow (\textit{Node} \mathrel{+\!\!\!\!\rightarrow} \mathbb{N}) \\
\hline
\forall\, f : \textit{Node} \rightarrow \mathbb{N};\ \textit{max} : \mathbb{N} \bullet \textit{incrementcounters}\ f = \\
\qquad \{\textit{node} : \textit{Node};\ n : \mathbb{N} \mid (\textit{node}, n) \in f \bullet \\
\qquad\qquad (\textit{node}, \mathbf{min}\{n + 1, \textit{NeighbourEmptyAg}\})\}
\end{array}
$$

The reset for all determined cells is straightforward.

$$\begin{array}{|l}
\textit{UpdateCounterDeterminedAg} \\
\Delta \textit{AgentCellAg} \\
\hline
\textit{type} = \textit{DeterminedAg} \\
\textit{counter} = \textit{counter}' + 1
\end{array}$$

The reset for stem cells depends on whether the cell is mature. In all cases the counters for the empty niche are updated.

$$\begin{array}{|l}
\textit{UpdateCounterStemAg} \\
\Delta \textit{AgentCellAg} \\
\hline
\textit{type} = \textit{StemAg} \\
\textit{counter} < \textit{CyclingPhaseAg} \Rightarrow \textit{counter} = \textit{counter}' + 1 \\
\textit{counter} = \textit{CyclingPhaseAg} \Rightarrow \textit{counter}' = \textit{CyclingPhaseAg} \\
\textit{nscounter}' = \\
\qquad (\textit{incrementcounters nscounter}) \oplus \\
\qquad\qquad (\{n : \textit{Node} \mid \textit{nstype } n \neq \textit{EmptyAg} \bullet (n, 0)\})
\end{array}$$

Request Division. Our agent-based approach to modelling forces us to consider what happens when two stem cells attempt to divide into the same location. In our model, we specify that when the internal counter reaches $\textit{CyclingPhaseAg}$, it signals to the environment the niche spaces that it is prepared to divide into.

Notice, that this approach is also agent-based in nature. Namely, the agent attempts to do something but the environment is a dynamic and uncertain one. From the perspective of a single cell with its limited sensory abilities the world is no longer deterministic like, it was in the cellular automata model, and not all attempts at action will be successful.

The agent-based model not only considers the nature of acting in a dynamic environment but also addresses issues such as the basic physical limitations of the stem cell niche in general. Once again, it's difficult to see how such issues can be considered, at least explicitly, with the cellular automata approach.

A stem cell agent that is ready to divide, signals to the environment those neighbours that have been empty for long enough, and so are able to receive the new cell. Of course the output may be empty.

$$\begin{array}{|l}
\textit{RequestDivision} \\
\textit{AgentCellAg} \\
\textit{possnodes}! : (\textit{AgentId} \times \mathbb{P}\, \textit{Node}) \\
\hline
\textit{counter} = \textit{CyclingPhaseAg} \\
\textit{possnodes}! = (\textit{id}, \{n : \textit{Node} \mid \textit{nscounter } n = \textit{NeighbourEmptyAg} \bullet n\})
\end{array}$$

The Environment Allocates Nodes for Division. The environment receives requests from cells to divide, and non-deterministically assigns those cells that can divide and those that have insufficient space around them. There are several safety properties that we can specify here:

1. all agents get a reply (first predicate)
2. no agent can be told to divide and not divide (second predicate)
3. no node ever has more than one agent dividing into it (third predicate)
4. cells only get to divide into a node they have requested (fourth predicate)
5. there is no remaining empty node that has been requested by any of the agents not-granted division (fifth predicate).

$$
\begin{array}{|l}
\hline
\textit{DetermineDivision} \\
\Xi \textit{AgentSystemStateAg} \\
\textit{requests?} : \textit{AgentId} \rightarrow (\mathbb{P}\,\textit{Node}) \\
\textit{divide!} : \textit{AgentId} \rightarrowtail\!\!\!\!\!\!\;+\; (\textit{Node} \times \textit{AgentId}) \\
\textit{nodivide!} : \mathbb{P}\,\textit{AgentId} \\
\hline
\mathsf{Let}\ \textit{cellsdividing} == \mathrm{dom}\,\textit{divide!} \bullet \\
\mathsf{Let}\ \textit{cellsnotdividing} == \textit{nodivide!} \bullet \\
\mathsf{Let}\ \textit{cellsrequesting} == \mathrm{dom}\,\textit{requests?} \bullet \\
\mathsf{Let}\ \textit{nodesreceiving} == \\
\qquad \{n : \textit{Node};\ \textit{id} : \textit{AgentId} \mid (n, \textit{id}) \in (\mathrm{ran}\,\textit{divide!}) \bullet n\} \bullet \\
\textit{cellsdividing} \cup \textit{cellsnotdividing} = \textit{cellsrequesting}\ \wedge \\
\textit{cellsdividing} \cap \textit{cellsnotdividing} = \{\}\ \wedge \\
\#\textit{cellsdividing} = \#\textit{nodesreceiving} \\
\hline
\end{array}
$$

2.4 Division and Determination

Cells that divide get told where they should divide into. We have two alternatives with the assignment of identifiers to the daughter cells. We can either give both daughters new identifiers, which is useful for tracking where they cells from, or the daughter cell which remains in the node of the previous cell keeps the id of its parent. We specify the first of these alternatives here.

$$
\begin{array}{|l}
\hline
\textit{AgentDivideAg} \\
\Delta \textit{AgentSystemStateAg} \\
\textit{parent?} : \textit{AgentId} \\
\textit{to?} : \textit{Node} \\
\textit{daughter1Id?}, \textit{daughter2Id?} : \textit{AgentId} \\
\hline
\mathsf{Let}\ \textit{cell} == (\mu\, a : \textit{AgentCellAg} \mid a.\textit{id} = \textit{parent?}) \bullet \\
\mathsf{Let}\ \textit{currentnode} == \textit{cells}^{\sim}\,\textit{cell} \bullet \\
\mathsf{Let}\ \textit{daughter1} == \\
\qquad\qquad (\mu\, \textit{ag} : \textit{InitAgentStemCellAg} \mid \textit{ag}.\textit{id} = \textit{daughter1Id?}) \bullet \\
\mathsf{Let}\ \textit{daughter2} == \\
\qquad\qquad (\mu\, \textit{ag} : \textit{InitAgentStemCellAg} \mid \textit{ag}.\textit{id} = \textit{daughter2Id?}) \bullet \\
\textit{cells}' = \textit{cells} \oplus \{(\textit{currentnode}, \textit{daughter1})\} \cup \{(\textit{to?}, \textit{daughter2})\} \\
\hline
\end{array}
$$

If the cell is not allowed to divide then id does nothing.

$$\begin{array}{|l}
\hline
\mathit{AgenNoDivideAg} \\
\Xi \mathit{AgentSystemStateAg} \\
\mathit{id}? : \mathit{AgentId} \\
\hline
\end{array}$$

Stem cells which have reached their cycle phase and which are surrounded by stem cells become determined.

$$\begin{array}{|l}
\hline
\mathit{AgentDeterminationAg} \\
\Delta \mathit{AgentSystemStateAg} \\
\mathit{cell}, \mathit{newcell} : \mathit{AgentCellAg} \\
\mathit{node} : \mathit{Node} \\
\hline
\mathit{node} = \mathit{cells}^{\sim}\, \mathit{cell} \\
\mathit{cell.type} = \mathit{StemAg} \\
\mathit{cell.counter} = \mathit{CyclingPhaseAg} \\
\mathrm{ran}\ \mathit{cell.nstype} = \{\mathit{StemAg}\} \\
\mathit{newcell.type} = \mathit{DeterminedAg} \\
\mathit{newcell.counter} = 0 \\
\mathit{cells}' = \mathit{cells} \oplus \{(\mathit{node}, \mathit{newcell})\} \\
\hline
\end{array}$$

3 Discussion

We have run hundreds of simulations of both the original CA model and of our agent recapitulation to check that the behaviours of our agent model has the same properties of the CA model. As we explained above, the agent model has allowed us to do is address the issues of biological implausibility.

It is interesting to note that allowing cells to split into all available spaces, i.e. up to four daughters, gives us the closest possible agent-based simulation match to the original CA models, however, any biologically plausibility we may have introduced would be negated by this. By limiting cell division to result in a maximum of at most two daughter cells we still maintain the integrity of the original cellular automata version.

In the next section we now explore how we have used and agent-based approach to extend one of the most sophisticated models of the stem cell niche that we have seen in the literature that proposes an innovative way of understanding how stem cell properties are maintained by the niche.

From a biological viewpoint the model of Agur et al. does not allow any reversibility or plasticity in the basic properties of cells. For example, once a cell has differentiated it cannot become a stem cell again. Moreover, once a cell has left the niche, it cannot return.

A recent example of an approach that uses a more sophisticated model and addresses these issues, is that of Markus Loeffer and Ingo Roeder at the University of Leipzig, who model hematopoietic stem cells using various, but limited, parameters including representing both the growth environment within the marrow, one particular stem cell niche, and the cycling status of the cell [8]. The ability of cells to both escape and re-enter the niche and to move between high and low niche affinities, referred to as within-tissue plasticity, is stochastically determined.

The validity of their model is demonstrated by the fact that it produces results in global behaviour of the system that match experimental laboratory observations. The point is that the larger patterns of system organization emerge from these few simple rules governing variations in niche-affinity and coordinated changes in cell cycle.

There is no doubt that Roeder's model is one of the most sophisticated ones that we have seen in the literature; it is formal, there is a simulation, it addresses key issues of self-organisation and much of the modelling has an agent-like quality to it. There are, however, a number of issues regarding this model that we have addressed by extending it using our agent framework. Most significantly, they use of a probability function to control the movement of cells between environments, and in the agent-view this is problematic. This probability is calculated from global information relating to the numbers of various cells in the system. Although it useful to assume access to this global information when developing the model of stem cell behaviour, no mechanism is known for how stem cells could have access to this information in real biological systems.

Space presents us to show our work here, but to summarise we have extended the Roeder model to produce an agent-based simulation that increases the biological intuition and plausibility of the model, and allows us to investigate emergence due to the subtle changes in micro-environmental effects for each cell. Modelling cells as agents responding autonomously to their local environment is much more fine grained than the previous model using equations to model cell transitions and allows for a much greater degree of sophistication in the possibilities of understanding how self-organisation actually takes place in the adult human body.

The main point is that an agent does not rely on getting information about the system state, in keeping with the reactive multi-agent systems approach, and we believe that this gives a more biologically plausible handle on how things might be working at the micro-environmental level.

We have extended the Roeder model to incorporate a model of space, albeit only in 2 dimensions so far, so that we can consider cell movement in more detail. We are particularly interested in experimenting with different shapes of niche to discover how these might affect the production or maintenance of stem cells and determined cells.

4 Concluding Remarks

It is perhaps worth noting that Roeder's model is similar in notion to Carriani's view of thermodynamic emergence [3]. It assumes that simple rules, i.e. the transition probability functions, can model complex behaviours of stem cells as they make their transition between niche and non-niche. The assumption is that complex behaviour can be understood by building models with simple behavioural rules that hide the complexities of the underlying interactions between many components, i.e. a top down approach to modelling.

By contrast, our model is more akin to Carriani's ideas of computational emergence. In this view, a series of simple rules gives rise to complex global behaviour, a bottom up approach if you like, we build simple models of agents and chemical diffusion the lead to the emergence complex system-wide behaviours.

We are currently extending our work by analysing other models and simulations using our formal methods and developing new implementations of these models using agents. We are also continuing to work with Theise to specify new models of his theories using our experiences of analysing and implementing other models of stem cell systems.

We are investigating ways of comparing outputs from our simulation runs, and looking at metrics for determining when one simulation can said to be similar or share the same emergent properties as another simulations. Formal methods have been very useful in that they are re-usable, directly relate to the implementation, and enable us to readily extend and agentify existing work.

From these case studies we can start to produce a kind of generic agent-based framework and simulation environment for modelling and simulating natural biological systems in 2 or 3 dimensions using an agent-based perspective. We believe modelling complex biological systems using an agent-based framework helps to ensure that models have biological plausibility and we also believe they are the most appropriate way of beginning to understand how complex self-organising behaviours occur in natural systems.

In this paper we have had several aims. First, we believe that recent medical evidence suggests that the way to understand how stem cells organise themselves in the body is as a self-organising system, whose global behaviour is an emergent quality of the massive number of interactions of cells with each other and of the environment of which they are a part. We claim. therefore, that the multi-agent system approach to modelling is the most suitable one for exploring means to simulate the behaviour of stem cells and from resulting simulations, suggest how tiny changes in individual stem cell behaviour might lead to disease at the global, and hence observable from an experimental perspective, system level. We have outlined the benefits of this approach by comparing it to a cellular automata approach in detail. Furthermore, we have aimed to demonstrate the pivotal role of formality not only in precision and clarity with modelling and in developing correct and consistent simulations, but as the foundation for a common conceptual framework in a multi-disciplinary project.

Acknowledgements

The team of collaborators in this project, entitled CELL, included the stem cell researcher Neil Theise, the artist but also the curator Peter Ride and the artist Jane Prophet both from the University of Westminster.

References

1. Z. Agur, Y. Daniel, and Y. Ginosar. The universal properties of stem cells as pinpointed by a simple discrete model. *Mathematical Biology*, 44:79–86, 2002.
2. R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.
3. P. Cariani. Emergence and artificial life. In C. Langton, C. Taylor, J. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 775–797, 1991.

4. M. d'Inverno and M. Luck. Development and application of a formal agent framework. In M. G. Hinchey and L. Shaoying, editors, *ICFEM'97: Proceedings of the First IEEE International Conference on Formal Engineering Methods*, pages 222–231. IEEE Computer Society, 1997.
5. M. d'Inverno and M. Luck. *Understanding Agent Systems (Second Edition).* Springer, 2004.
6. M. d'Inverno, N. D. Theise, and J. Prophet. Mathematical modelling of stem cells: a complexity primer for the stem cell biologist. In Christopher Potten, Jim Watson, Robert Clarke, and Andrew Renehan, editors, *Tissue Stem Cells: Biology and Applications*. Marcel Dekker, to appear, 2004.
7. D. S. Krause, N. D. Theise, M. I. Collector, O. Henegariu, S. Hwang, R. Gardner, S. Neutzel, and S. J. Sharkis. Multi-organ, multi-lineage engraftment by a single bone marrow-derived stem cell. *Cell*, 105:369–77, 2001.
8. M. Loeffler and I. Roeder. Tissue stem cells: definition, plasticity, heterogeneity, self-organization and models – a conceptual approach. *Cells Tissues Organs*, 171:8–26, 2002.
9. M. Luck and M. d'Inverno. A conceptual framework for agent definition and development. *The Computer Journal*, 44(1):1–20, 2001.
10. Michael Luck, Ronald Ashri, and Mark d'Inverno. *Agent-Based Software Development*. Artech House, 2004.
11. N. Nicola and G. Johnson. The production of committed hemopoietic colony-forming cells from multipotential precursor cells in vitro. *Blood*, 60:1019–1029, 1982.
12. J. Novak and C. Stewart. Stochastic versus deterministic in haemopoiesis: what is what? *Br J Haematol*, 60:527–529, 1991.
13. M. Ogawa. Stochastic model revisited. *International Journal Hematology*, 69:2–6, 1999.
14. I. Roeder. Dynamical modelling of hematopoietic stem cell organisation. *Ph.D. Dissertation Leipzig University*, 2003.
15. M. Spivey. *The Z Notation (second edition).* Prentice Hall International: Hemel Hempstead, England, 1992.
16. N. D. Theise. New principles of cell plasticity. *C R Biologies*, 325:1039–1043, 2003.
17. N. D. Theise and M. d'Inverno. Understanding cell lineages as complex adaptive systems. *Blood, Cells, Molecules and Diseases*, 32:17–20, 2003.
18. N. D. Theise and D. S. Krause. Toward a new paradigm of cell plasticity. *Leukemia*, 16:542–548, 2002.
19. I Thornley, R. Sutherland, R. Wynn, R. Nayar, L. Sung, G. Corpus, T. Kiss, J. Lipton, F. Doyle, J. Saunders, S. Kamel-Reid, M. Freedman, and H. Messner. Early hematopoietic reconstitution after clinical stem cell transplantation: evidence for stochastic stem cell behavior and limited acceleration in telomere loss. *Blood*, 99:2837–96, 2003.
20. J. Till, E. Mcculloch, and L. Siminovitch. A stochastic model of stem cell proliferation, based on the growth of spleen colony-forming cells. *Proc Natl Acad Sci USA*, 51:29–36, 1964.
21. J. Trentin. Influence of hematopoietic organ stroma (hematopoieticinductive microenvironment) on stem cell differentiation. *Gordon, A.S. (editor), Volume 1, Appleton-Century-Crofts, New York*, pages 161–168, 1970.
22. S. Viswanathan and P. Zandstra. Toward predictive models of stem cell fate. *Cryotechnology Review*, 41(2):1–31, 2004.

Ambient Cognitive Environments and the Distributed Synthesis of Visual Ambiences

Guillaume Bour, Guillaume Hutzler, and Bernard Gortais

Computer Science Methods Laboratory, UMR 8042, CNRS / Evry-Val d'Essonne University, 523 Place des Terrasses, 91000 Evry, France
hutzler@lami.univ-evry.fr, guillaume.bour@free.fr, bernard.gortais@lip6.fr

Abstract. One of the current trends in computer science leads to the design of computing organizations based on the activity of a multitude of tiny cheap decentralized computing entities. Whether these chips are integrated into paintings or disseminated in open environments like dust, the fundamental problem lies in their cooperative operation so that global functions are obtained collectively. In this paper, we address the issue of the creation of visual ambiences based on the coordinated activity of computing entities. These entities are distributed randomly on a 2D canvas and can only change their own color and perceive their immediate neighbors.

1 Introduction

It is a fact that research on *ubiquitous computing*, since Mark Weiser coined the term in 1988, has developed very rapidly [2], [3]. It is especially true for the last two or three years with the explosion of mobile telephony, PDAs, wireless networks, etc. Ubiquitous computing is associated to the disappearance of computers, not because they're not there anymore, but because they become invisible. But it's not because we can't see them anymore that we can't interact with them. The question of interaction with ubiquitous systems has not really been raised as such. What is studied is the interaction with mobile devices such as PDAs but what about the interaction with the "societies" of computing entities that will "live" and develop in our walls, objects, clothes, etc.? This is an almost sociological question and we argue, with others [14], that it could be studied efficiently using the computerized concepts and tools that are interested in the sociological aspects of computing, namely multi-agent systems.

Our approach consists in considering this interaction as a multimodal dialogue between a human user and his(her) physical environment. We develop this approach in a project called *DanCE with $(MA)^2$CHInE*[1], in which we consider the environment as being populated with dozens of physical communicating objects. Each of these communicating objects is characterized by limited capabilities for the treatment of information, the communication with others, and the interaction with their physical environment. The problem can then be reformulated as a problem of building decentralized cognitive systems, which we call *Ambient Cognitive Environments* (ACEs). To build such cognitive environments, one has to address the three main

[1] Dynamic Ambient Cognitive Environments with Multi-Agent Multimodal and Adaptive Computer-Human Interaction Engine.

processes that are typical of any situated cognitive system: perception, decision, action. However, each of these processes has to be handled in a decentralized way: we treat perception as a sensor fusion problem, decision as a distributed consensus reaching problem and action as a distributed coordination problem.

In this paper, we develop only the latter problem of action and we focus more precisely on the production of visual ambiences. In the meantime, we explain the abstractions that will allow us to extend the model to other kinds of expression such as music, choreography, etc. This model is based on an analytical approach to different artistic domains. In each of these domains, the analysis must lead to a description of the corresponding expression (visual, musical, choreographic, etc.) in terms of qualitative pairs such as cold/warm, quiet/loud, slow/fast, etc. These pairs form together an ontology that the multimedia production system should know and that it should be able to manipulate so as to express chosen emotions.

The aim is to be able to give instructions to the system using this ontology. The difficulty is then for the system to translate a given order into a coordinated activation of distributed colored cells. These cells are distributed randomly on a 2D canvas in a way which is similar as in the works on amorphous computing [1], [9], [10]. Individual cells can change their color and perceive other cells in their immediate vicinity but they can also move. Specific algorithms have therefore to be proposed so that all the cells in the canvas can collectively express specific colored contrasts or spatial structures. Such graphical composition primitives have finally to be composed in a way that preserves the individual properties of each.

The paper is organized as follows: in the next section, we elaborate some more on Ambient Cognitive Environments. We show in section 3 how the analysis of visual expression allows to propose a grid relating emotions with structural properties of pictures. We then show in section 4 how this can be expressed in a decentralized way by elementary colored cells.

2 Ambient Cognitive Environments

The work presented in this paper is part of a larger project in which the objective is to identify the right concepts and develop the corresponding technical tools to dynamically organize distributed sets of computing entities as cognitive systems. This is what we call ***Ambient Cognitive Environments*** (ACEs). The aim is that these high-tech environments become sensible to the people that live in them. The aim is not to be intrusive and spy the movements of these people, but to become aware of their emotional dispositions and adapt accordingly. The environment adapt by producing visual and sonorous ambiences that are calm when people are calm, or that become calm when children get too excited, or that become suddenly “flashy” and buzzing when people are too calm, etc. But not only the ambience may be adapted: specific actions may be done using motorized objects; specific displays could be produced on the walls, on clothes, either to establish a contact or to convey some information; finally, electronic devices such as mobile phones, PDAs, MP3 players, etc. could be used to send focalized audio or visual messages to one person.

Four important aspects, we believe, characterize these ACEs: first, people interacting with ACEs shouldn’t have any technical job to do to make them run, hence the automatic configuration of such environments depending on available sensors, effectors, computing resources, etc.; secondly, people shouldn't wear specific equipment to interact with ACEs, hence the focus on body capture techniques that rely or low-cost cameras, without any constraints on the body of the person; thirdly the interaction should be multimodal, using interaction modalities that people are used to, hence the focus on multimodal languages to analyze the performance of a dancer and the response of the system; finally, ACEs should be able to learn to adapt their responses to specific people, with specific ways of expressing emotions, and with specific sensibilities to visual and sonorous environments, hence the central importance of machine learning techniques in the project.

We may summarize all of this as the fact that the interface should not be more visible than the computers themselves. If computers are disappearing, the interface should also become as discrete and as natural to use as possible. We could finally imagine that these kind of environments may adapt to people with either perceptive or motor disabilities, by choosing dynamically the right modalities to establish a communication with them.

In order to be able to combine multiple modalities, we need to use a level of representation such that these different modalities can be described in a homogeneous way and compared with one another. We distinguish four levels of abstraction in the characterization of the behavior of the user: *raw data* are acquired directly by the various sensors; *primitives* correspond to quantitative measures, obtained by the processing of raw data (e.g. position, speed, etc.) ; *qualities* correspond to a subjective characterization of behavior by a set of pairs of terms (e.g. slow/fast, warm/cold, etc.); *emotions* finally correspond to more general terms used to characterize the global ambience of a situation (e.g. sadness, calm, liveliness, etc.). Although ill-defined, the latter notion of emotions correspond to the intuitive concept, based both on cognitive and physical reactions to some situations [11]. By defining these levels, the objective is twofold: first, to maintain a multilevel representation in order to allow an incremental analysis of the behavior; second, to be able to compare the various methods of capture, and therefore the expressions of the interlocutor, by using a representation that is abstract enough (*qualities* or *emotions*).

3 Analytical Study of Visual Expression

In this paper, we focus more specifically on the automatic generation of visual ambiences. This ambience is not meant to reflect the emotional state of the user [12] but rather to inspire chosen feelings to the spectator. Our approach is based on the hypothesis that these feelings rely, for some part, on the composition of contrasts and graphical structures. To do this, we first need to present some general considerations about the analytical study of visual expression. Any picture, either a photography or a painting, has an emotional content. Depending on the cultural and historical context, we perceive pictures with different codes, that make us feel various emotions such as happiness, sadness, calm, serenity, etc. This also depends on the receptiveness of each individual person but we can consider that the interpretation code is largely shared inside a given culture. A sunset over the sea for example (see figure 1) generally produces a tragic effect and inspires feelings of calm and serenity. This common emotional answer to pictures has been analyzed and codified at the beginning of the XXth century by artists such as W. Kandinsky [6][7] and P. Klee [8]. Kandinsky, in particular, tried to identify the role of shapes, colors, contrasts in the production of emotions. However, he hardly said anything about the interactions between these elementary components because of the complexity of this study.

3.1 Spatial Structures and Contrasts

A picture can be decomposed into several zones, each of which can be associated to distinct “tensions”. While the top of a picture symbolizes lightness, ascension, freedom, the bottom symbolizes heaviness and constraint. In addition, objects in the picture are organized along abstract structuring lines that express the dynamics. While horizontal and vertical lines symbolize calm and rest, diagonals express movements.

Finally, contrasts express oppositions between graphical objects that make them reinforce each other. Each individual characteristics of the objects, such as color, shape, size, etc., may give rise to a corresponding contrast. For color only, there are seven identified contrasts, which are based on the properties of colors: contrast of pure hue, value contrast, intensity contrast, complementary contrast, temperature contrast, size contrast and finally simultaneous contrasts. Properties of the shapes can also be used to produce size contrasts (small objects vs. large objects) or shape contrasts (symmetrical vs. unsymmetrical objects). Finally, if pictures are animated, contrasts can be built using the properties of movement such as direction, speed or rhythm.

3.2 Composition of Elementary Properties

Only through the composition of chosen elementary spatial structures and contrasts can more complex emotions be expressed. Table 1 is an attempt to summarize basic correspondences that we may establish, in western culture, between the composition of pictures, expressed in terms of graphical structural and contrasts, and the emotional content of the picture.

Table 1. Correspondences between the composition of pictures (in terms of spatial structure and contrasts) and their emotional content

Contrasts	dynamic	static	value	temperature	complementary	pure hue	size	intensity	simultaneous	top-bottom position	left-right position	shape	size
Emotions	**Structure**		**Colors**										
			Graphical objects										
happy	o					o						o	
calm		o						o					
serene				o						o			
sad			o					o					
restless	o	o				o						o	
tragic			o				o			o			o

If we come back to the example of a sunset over the sea (see figure 1), we can analyze it in the following way:

- the picture shows an horizontal structure, separating the clouds, the sky, the sea;
- several contrasts can be identified
  - a value contrast between the dark sea and lighter clouds and sky
  - a temperature contrast between the sun and surrounding sky, which are very warm (yellow orange), and other regions of the picture, with colder colors
  - a size contrast between the small sun and the large regions around (sea, clouds)
  - an intensity contrast centered on the sun.

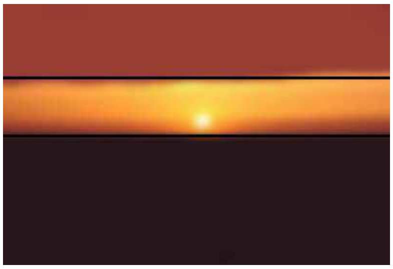


Fig. 1. Sunset over the sea. The picture can be analyzed in terms of contrasts and a graphical structure

Consultation of table 1 thus tells us that the emotional qualities of this picture are the following:

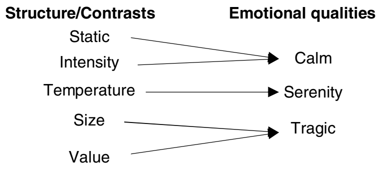


Fig. 2. Correspondence between structural properties, contrasts and emotional qualities for the picture shown in figure 1

3.3 Generic Modeling of Contrasts

Although contrasts can produce very different visual results, most of them are based on the same principles. This is why we propose a generic model of the way a contrast can be built by the association of graphical objects.

1. a given contrast is built upon an opposition of graphical objects with respect to a specific graphical property (color, shape, size, etc.);
2. objects in the picture are divided into distinct groups (generally two); each of them has its own value for the considered property: in a value contrast for example, some of the objects are characterized by a light value, others by a dark value;
3. there's a quantitative imbalance between the groups: the objects of one group are more numerous than the others;

4. for some contrasts, there's also a spatial imbalance between the groups: objects of the preponderant group are distributed on the whole surface while the objects of other groups are distributed along the main structuring lines of the picture.

4 A Multi-agent Composition

We have defined the theoretical background of the work from a graphical point of view. We can now explain how it can be implemented as a multi-agent system in which emotional qualities of pictures will be obtained by the coordinated activation of elementary autonomous colored cells. By choosing such a decentralized approach, the aim is to provide methods and algorithms that may be used on decentralized displays. Today's displays rely on LCD or plasma technology. Tomorrow's ones may perfectly well rely on tiny processors integrated in paintings that may change the color of the painting in their vicinity, thus composing together displays as big as entire walls [1]. In addition, the proposed approach can be seen more generally as a way of spatially structuring entire networks of processors, which may be very valuable in contexts such as smart dust or sensor networks.

Our objective is definitely not to reproduce specific pictures or patterns, but to provide the cells with the capability to manipulate contrasts and spatial structuring as a mean to produce chosen emotional qualities. We based our work on the following assumptions:

- algorithms should function with irregular 2D distribution of computing cells, either static or dynamical (somewhat similar to the distribution that is used in the works on amorphous computing);
- convergence of the algorithms should be fast so that the generation of pictures is also fast;
- since the very notion of contrast is very general and can be instantiated in different ways, the generation of contrasts by the system should be as generic as possible (see paragraph 3.3 above)
- finally, although the perception of emotional qualities is quite general inside a given culture, individual variations evidently exist and the algorithms must allow the integration of machine learning techniques.

4.1 Composition of Elementary Behaviors

The challenge is now to implement the model presented in paragraph 3.3. with a multi-agent approach. To do this, we chose to rely on a modular approach, decomposing the overall problem into separate concerns. Step 2 in the model (allocation into groups) can be realized quite easily since it doesn't involve any coordination between the agents. The agents will thus realize it independently of each other. Step 3 (quantitative imbalance), on the contrary will need the coordination of all the agents in the picture, with no possible centralization. This will require the following sub-steps:

3.1. collective choice of the dominant group;
3.2. collective count of each group's population;
3.3. migrations of agents between the groups so as to obtain a given ratio;

Similarly, step 4 (spatial imbalance) will require the following sub-steps:

4.1. collective choice of the dominant group;
4.2. homogeneous distribution of agents of the dominant group across the picture and distribution of agents of the other groups along the structuring lines of the picture.

Steps 3.1 and 4.1 are identical, which finally produces the composition schema shown in figure 3.

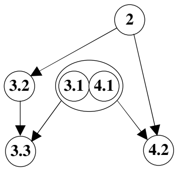


Fig. 3. Composition schema of sub-steps for the construction of contrasts: arrows correspond to functional dependences between the modules

4.2 Step 2: Concentration into Value Intervals

For a given contrast, the opposition between the graphical objects is based on different values for a specific graphical property, for example hue. For a temperature contrast, some of the agents will adopt a warm hue (yellow, orange, red) while others will adopt a colder hue (blue, green). This doesn't mean that all the agents of a group will adopt a given value and that all the agents of the other group will have another fixed value. This rather mean that the agents of one group will have their hue distributed in a given interval of values (340-20 in the chromatic circle) and that the agents of the other group will choose their hue in another interval (210-260 in the chromatic circle). We made the choice to represent all possible properties as intervals of numerical values, inside which we can choose 2 opposing intervals and have the agents distribute themselves into these intervals. The intervals of values can be in direct correlation with the modeled properties (as it is the case with hue) but they can also be abstract descriptions of some properties (as it is the case with a symmetry parameter, which is not directly quantifiable but which can be measured and associated to an abstract scale ranging from 0 to 100). Some of the intervals can also be cyclic as is shown in figure 4.

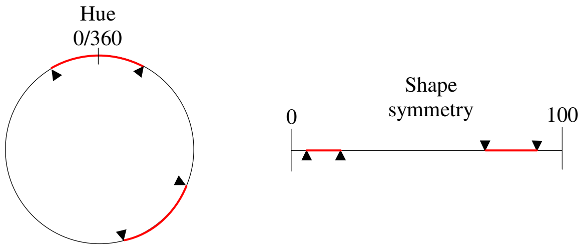


Fig. 4. Value intervals representation, compatible with circular or linear scales

Each of the agents being initially in a random state with respect to the property chosen for the contrast, the agents must evolve to come closer to the specified intervals. They do so at each simulation step with the algorithm shown below.

```
# Activation
∀ interval i, compute $d_i$ = distance to i
choose interval j so that $d_j$ = $Min_i$ $d_i$
change property x towards interval j
```

Figure 5 shows the result of the concentration algorithm for the temperature contrast. Each colored square corresponds to an agent. At the beginning of the simulation, each agent is in an undetermined state. Agents rapidly change their hue towards the two intervals (340-20 and 210-260).

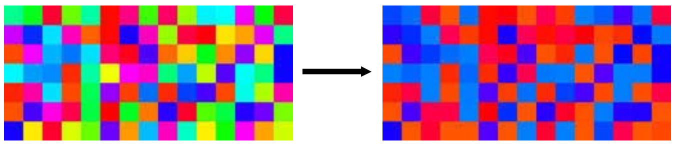


Fig. 5. Concentration into separate intervals of hue for temperature contrast

4.3 Step 3.1: Choice of the Dominant Group

This step is meant to choose the group that will become dominant. This collective choice has to be random and equiprobable. The problem of the choice, or voting, has been studied by D. Schreiber [13] and G. Weiss [15]. In the model of Schreiber, agents spatially organize according to affinities and move to form coalitions. Since the position of the agents is taken into account in some of the contrasts that we want to realize, this was not satisfactory. In the protocols proposed by Weiss, agents order possible solutions depending on their individual preferences. However, this implies a lot of communications since individual votes have to be collected, compared, and diffused back to all the agents.

In our problem, the final choice isn't important as such. We don't care about satisfying the initial choices of agents, we only care about obtaining a single final choice. The solution that we propose consists in aggregating incrementally the votes of closest neighbors. After making a random initial choice, the behavior for each agent at each simulation step is the following:

```
# Initialization
choice = random (1..groups_nb)
weight = 1

# Activation
for c = 1 to groups_nb do
  neighbors_weight[c] = sum neighbors with (choice = c)
done
choice = i so that neighbors_weight[i] == $max_i$(neighbors_weight[i])
weight = neighbors_weight[choice]
```

Evaluation

This evaluation is not a formal proof of convergence of the algorithm but gives indications about its quality. The two criteria were the quality of the random distribution and the speed of convergence. To this end, agents were ask to make their choice between four different colors (red, blue, green, yellow). The simulation has been done with 100 moving agents in a 550x550 pixels space with a 100 pixels perception distance. 100 runs of the simulation have been done.

Table 2 shows the distribution of the 4 possible choices. We can see that the number of occurrences for each choice is very close to the mean value.

Table 2. Distribution of choices for 100 runs

Choice	Red	Blue	Green	Yellow	Total	Mean	Mean deviation
Results	22	30	26	22	100	25	3

Figure 6 below shows, for each convergence time expressed in number of simulation cycles, the number of runs that have converged in that time. One can see that for all the runs, the convergence time is comprised between 2 and 17 cycles, with a mean of 9,1 cycles.

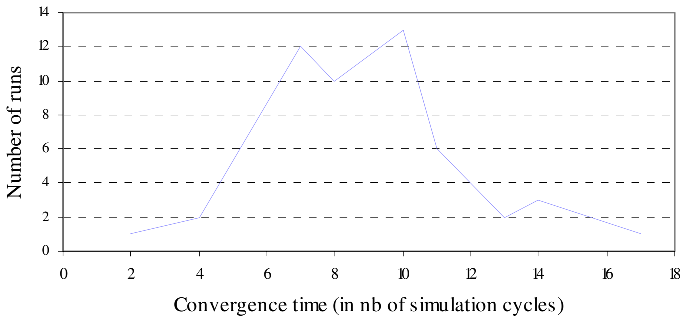


Fig. 6. Number of runs vs. convergence time

4.4 Step 3.2: Count of Groups Population

Quantitative imbalance requires that we assess the relative size of the different groups. To this end, we chose to count the number of agents in each group.

Our solution is inspired by the BFS algorithm that computes the diameter of a network (the distance between the most distant nodes) by building a covering tree. The difference is that we don't have any predetermined topology (the connectivity between the agents isn't static because they can move across the environment). The solution is also adaptive because the count is updated when agents change from one group to another. The algorithm is shown next page.

```
# Initialization
each agent gets a token
each agent propagates a "presence" stimulus

# Activation
1. Aggregation of agents into associations; the "leader" gets all the
   tokens of the other agents in the association
2. Fusion of associations with one another
3. Diffusion of the result (total number of tokens) to all the agents
```

The algorithm proceeds in three steps:

1. The agents form associations, each of which has got a "leader". The latter centralizes all the tokens of the association. This step is inspired by the Clubs algorithm described in [9].
2. Once the associations are formed, they try to merge:
   - ▷ agents at the border between two associations propagate a gradient towards the leader. The gradient contains the information of the distance to the border, incremented at each agent jump. The gradient diffusion method is described in [10].
   - ▷ when the leader perceives the gradient, it transfers its tokens to its neighbors that is closest to the border (the one that diffused the gradient with the smallest distance). This agent becomes the new leader of the association. The tokens and the leadership thus move from agent to agent towards one of the border of the association.
   - ▷ when two leaders are close enough, they can merge. One of them collects the tokens of both and the corresponding associations become merged.
3. When all the associations have merged, only one leader remains that has collected all the tokens of all the agents of the simulation. He can then diffuse the result to the other agents using gradient diffusion.

Figure 7 shows successive steps in the merging of associations. The first picture corresponds to the state of the system after the constitution of associations. Each subsequent picture corresponds to the fusion of two associations.

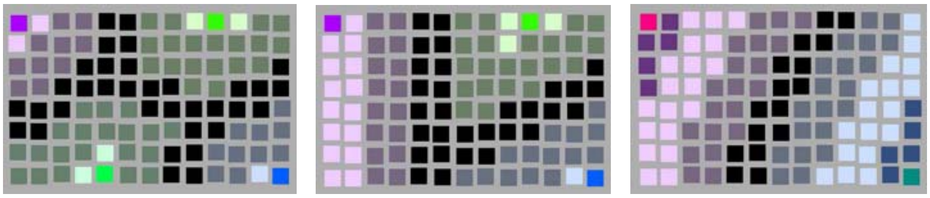


Fig. 7. Steps in the merging of associations: black agents corresponds to borders; leaders are lighter

Since the leader of the last association has collected the tokens of all the agents, he's got as many tokens as the number of agents in the simulation. The algorithm thus counts the agents. When constructing contrasts, we can thus evaluate the number of agents inside each of the different groups. When agents are distributed inside several groups (two for a temperature contrast), the algorithm has to be executed in each group. Furthermore, we will explain in the next paragraph that agents will be able to change group. When this arrives, the count must be dynamically updated. To this end, the migrating agent gets a negative token (-1) for the group that it leaves, and a positive token (+1) for the group it joins. As the algorithm is continuously executed, it converges very rapidly towards a new result.

Evaluation

We evaluated the algorithm in the same conditions as the choice algorithm in order to assess the time necessary to converge towards a global results. Each run is stopped when all the agents have received the correct count.

Table 3. Time to converge towards a global count, diffused to all the agents of the simulation

Duration	Arithmetic mean	Min	Max	Standard deviation
Results	18,8	14	27	3,08

Although adaptive, the algorithm still presents some weaknesses:

- if a leader fails, the tokens that it was responsible for are lost. This may be a problem for amorphous computing in which agents correspond to chips and are thus exposed to potential failures;
- convergence is slower when agents move (associations are less stable) or when they frequently change groups.

4.5 Step 3.3: Quantitative Imbalance

Once the agents have chosen the group that will be dominant and they have evaluated the respective size of all the groups, we can adjust the ratio between the groups. This is done by having agents migrate from one group to the other. It is necessary to specify beforehand the desired ratio between the different groups (e.g. 10%-90%). The algorithm, for each agent at each simulation cycle is the following:

```
# Activation
if total[my_group]/sum(total[]) > percentage[my_group]
  token[my_group] -= 1;
  my_group = another_group
  token[my_group] += 1;
end_if
```

Figure 8 shows the quantitative imbalance for the temperature contrast. The desired ratio is 20% for warm-colored agents and 80% for cold-colored agents.

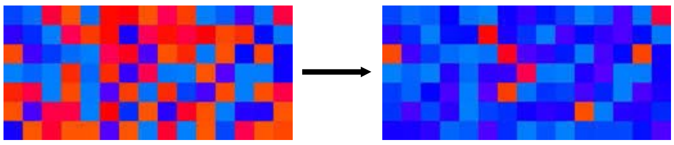


Fig. 8. Quantitative imbalance for temperature contrast. Desired ratio of 20%-80%

4.6 Step 4.2: Qualitative Imbalance

The role of this final step is to organize the graphical elements spatially. Our approach consists in positioning attracting agents that propagate gradients in their vicinity. These gradients are meant to structure the distribution of agents from the non-dominant groups. If we have only one attracting agent, the result is a spot of agents that contrast with a homogeneous background (see figure 9).

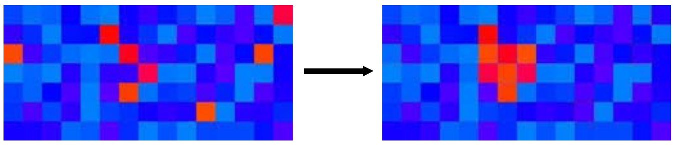


Fig. 9. Single attractive agent

To obtain more complex structures, the approach is inspired by composition rules used by painters. Each Border of the picture is divided into three thirds of four quarters. The points so defined can be joined together, which creates structuring lines. These structuring lines can either be static (horizontals and verticals) or dynamic (diagonals). Such lines will be generated by placing "anchor" agents along the borders at the dividing points and to make them propagate linear gradients. The gradients are characterized by the angle φ that they make with the horizontal. Whether we wish to obtain static or dynamic structures, the probability to generate anchor agents with φ equals to 0° (horizontal line) or 90° (vertical lines) will be more or less high (see figure 10).

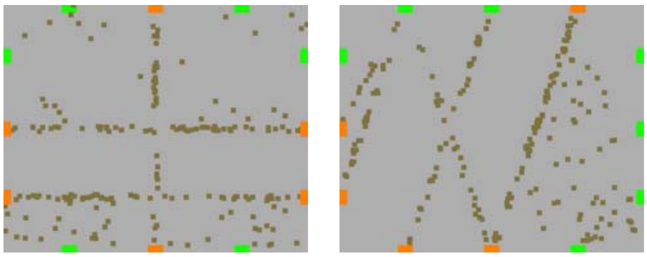


Fig. 10. Static (*left*) and dynamic (*right*) structuring of the picture using either horizontals and verticals or diagonals

Anchor agents can then be distributed along these structuring lines. When they activate, they propagate gradients that attract agents from the non-dominant groups and make them group in localized spots around them. In turn, the spots restitute for the viewer the feeling of "virtual" lines that organize the composition. Figure 11 shows a first attempt to organize the picture according to such principles. In this example, the agents did actually move but the same can be obtained if agents are in a fixed position: a virtual movement can be obtained by exchanging the properties of two neighboring agents.

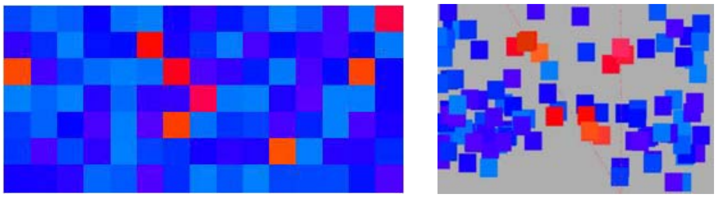


Fig. 11. Dynamical structuring of the picture along composition lines

4.7 Exceptions to the Generic Model

Two specific color contrasts (simultaneous and intensity contrasts) did not fit well into our generic model. The simultaneous contrast corresponds to the association of a given color to its gray component (i.e. the color we would obtain by changing the picture into grayscale). When viewing such a contrast, we tend to see the complementary color at the boundary between the color and the gray. The intensity contrast corresponds to the association of saturated and unsaturated colors (the latter must prevail) in the picture.

For these two contrasts, the solution we propose relies on the use of a gradient that is propagated around the center of the contrast. This gradient is provided with a *distance* information that is propagated and incremented from one agent to the next. The distance is equal to 0 at the center and is incremented as we move away from it. The distance is then put into correspondence with the characteristics that is involved in the contrast: for the simultaneous contrast, the color of the agents is unsaturated until a given distance from the center; for the intensity contrast, the saturation of the color of the agents decreases as a function of the distance from the center (see figure 12).

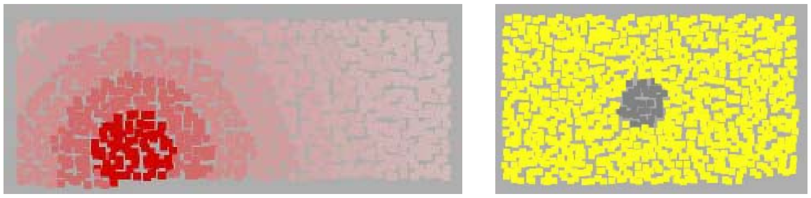


Fig. 12. Intensity contrast (left) and simultaneous contrast (right)

4.8 Composition of Several Contrasts

Once isolated contrasts can be obtained, we have to address the issue of composing distinct contrasts with one another. When contrasts are orthogonal (they rely on distinct properties), they can be combined either by aligning the two contrasts or by generating them in a disjoint way.

In the case of aligned contrasts, the first contrast (we call it the “master” contrast) is built by using the technique that we described in this paper. The second one (we call it the “slave” contrast) is then built by executing only step 2 (see paragraph 4.2) on a different characteristics of the agents. For example, after building a temperature contrast, we can simply superimpose a value contrast by making warm colors brighter and cold colors darker (see figure 13). In the case of disjoint contrasts, the whole algorithm has to be run twice, once for each contrast (see figure 13).

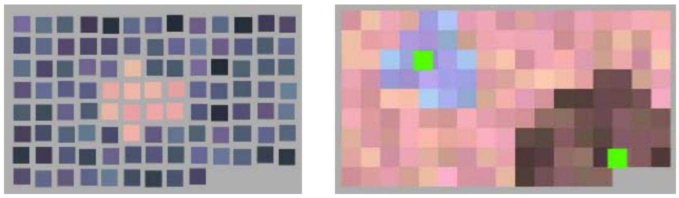


Fig. 13. Aligned contrasts (left) and disjoint contrasts (right)

4.9 Parameterization of the Model

In order to be interesting, the result has to be variable and changing, and it has to be adapted to the user. This can be done using several strategies:

- in step 2, the intervals that we choose to define the contrast will greatly influence the result that we obtain. Indeed, the contrast will be stronger when the distance between the intervals is bigger (see for example figure 14);
- a stronger contrast will also be obtained by using a bigger ratio between the different groups;
- finally, a user may change the correspondence shown in table 1 between emotions and pictural means needed to express them.

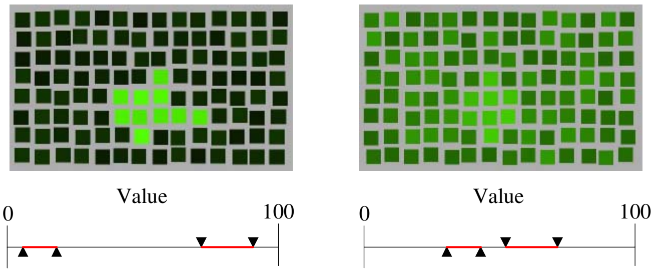


Fig. 14. Different value contrasts obtained by varying the intervals of the 2 groups

5 Conclusion

We presented in this paper a global and coherent approach to the creation of visual ambiences, based on the use of contrasts and spatial structuring of pictures. Inspired on the one hand by the analytical works of painters like Kandinsky and Klee, and on the other hand by researches on amorphous computing, our work demonstrates the feasibility of a decentralized approach. In particular, we presented a generic and modular model for the creation of contrasts. This model relies on decentralized algorithms that implement collective choices and counts, which allows to distribute agents in separate groups and to control the relative abundance of the groups. We showed that these algorithms converged fast towards global solutions, which may lead to new applications for amorphous computing. In particular, we showed how a spatial structuring of processor networks may be obtained by using contrasts and composition structuring rules. In addition, this model can easily be adapted to different users by changing some parameters, either by hand or by using machine learning techniques.

Future research directions will explore this parameterization problem in a more detailed and systematic way. We will use genetic algorithms on the one hand to produce various system behaviors; we will define validation protocols on the other hand in order to be able to assess the efficiency of the system, not in terms of speed of convergence but in terms of emotional qualities expressed by the system.

References

[1] Abelson H., Allen D., Coore D., Hanson C., Homsy G., Knight T., Nagpal R., Rauch E., Sussman G., and Weiss R.. "Amorphous computing", in *Communications of the ACM*, 43(5), May 2000.

[2] Borriello G. and Holmquist L. E. eds., *UbiComp 2002*, Springer, Berlin, 202.

[3] Finin T., Maamar Z. eds, *Workshop 16 - Ubiquitous agents on embedded, wearable and mobile devices*, AAMAS 2002.

[4] Hutzler G., Gortais B., Drogoul A., "The Garden of Chances: a Visual Ecosystem", in *Leonardo*, Vol. 33, Issue 3, pp. 101-106, April 2000, International Society for the Arts, Sciences and Technology, MIT Press.

[5] Hutzler G., Gortais B. and Orlarey Y., "Mutations: Plastic and Musical Improvisation by Distributed Agents", in *World Multiconference on Systemics, Cybernetics and Informatics 2001*, N. Callaos, X. Zong, C. Vergez and J. R. Peleaz eds, pp. 380-385, Orlando (Florida, USA), 2001

[6] Kandinsky W., *Concerning the Spiritual in Art*, Dover Publications, 1977.

[7] Kandinsky W., *Point and Line to Plane*, Dover Publications, 1979.

[8] Klee P., *Paul Klee on Modern Art*, Faber & Faber, 1985.

[9] Nagpal R. and Coore D., "An Algorithm for Group Formation and Maximal Independent Set in an Amorphous Computer", AI Memo 1626, MIT, 1998.

[10] Nagpal R., "Programmable Self-Assembly Using Biologically-Inspired Multiagent Control", in *AAMAS 2002*, pp. 418-425, Bologna (Italy), 2002.

[11] Picard R., *Affective Computing*, The MIT Press, Cambridge MA, 1997.

[12] Schemat S., "Virtual Emotion", in *Ars Electronica, Artificial Life – Genetic Art*, Linz, 1993 (http://www.aec.at/en/archive_files/19931/E1993_230.pdf).

[13] Schreiber D., "The Emergence of Parties: an Agent-Based Simulation", in *Annual Meeting of Midwestern Political Science Association*, Chicago, Illinois, 2000.

[14] Servat D., Drogoul A., "Combining amorphous computing and reactive agent-based systems: a paradigm for pervasive intelligence?", in *AAMAS 2002*, pp. 441-448, Bologna (Italy), 2002

[15] Weiss G., *Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence*, The MIT Press, 1999.

Using the Experimental Method to Produce Reliable Self-organised Systems

Bruce Edmonds

Centre for Policy Modelling,
Manchester Metropolitan University
cfpm.org/~bruce

Abstract. The 'engineering' and 'adaptive' approaches to system production are distinguished. It is argued that producing reliable self-organised software systems (SOSS) will necessarily involve considerable use of adaptive approaches. A class of apparently simple multi-agent systems is defined, which however has all the power of a Turing machine, and hence is beyond formal specification and design methods (in general). It is then shown that such systems can be evolved to perform simple tasks. This highlights how we may be faced with systems whose workings we have not wholly designed and hence that we will have to treat them more as natural science treat the systems it encounters, namely using the classic experimental method. An example is briefly discussed. A system for annotating such systems with hypotheses, and conditions of application is proposed that would be a natural extension of current methods of open source code development.

1 Introduction

Zambonelli and Van Parunak (in parallel with others) have pointed out that, increasingly, a different *kind* of computer system will be required if we are to meet many of society's needs [21] and these are starting to be developed. In this paper I go further[1] and argue that in parallel with different *kinds of system* we will need a *different kind of approach* to producing such systems – an approach which places more emphasis on natural scientific approaches than has been usual in multi-agent systems. In other words, that design and engineering (in a sense I will make clear) must make more room for adaptation and experiment. As should become evident, I am not advocating the abandonment of design – far from it – but am suggesting a better balance.

I start by distinguishing what I call the 'engineering' and 'adaptation' approaches and their how they are applied (both separately and together). I then discuss some of the limitations of the engineering method by considering an apparently simple class of MAS that nonetheless is, in general, intractable to methodical and effective design methods (the limitations of the adaptation method being fairly obvious). In contrast I

[1] To be clear, it is not that Zambonelli and van Dyke in [21] don't see a need for a change in method as well as the change in system type, but that they do not see such a need to depart from the engineering approach to the extent I am suggesting.

show that these systems can be adapted to serve defined (albeit simple), purposes using an evolutionary algorithm. These two sections lead on to the conclusion that we will necessarily have to develop and deploy systems for which there is no complete understanding inferrable from its design. Such systems (and many others) will have to be understood using the same techniques that we use with other 'ready-made' systems in the natural world: by hypothesis and experiment. I then sketch how such a natural science of self-organised systems might be used to achieve fallible but high levels of reliability and (relatively) safe system reuse. I end by giving a short example to illustrate this before I conclude.

2 Two Approaches for Obtaining Useful Systems

There are two basic ways of getting a useful system: by designing and then implementing it, i.e. to construct it (what I will call the "engineering approach"); or by taking some existing system and then manipulating it until it is good enough (what I will call the "adaptive approach"). I briefly explain these approaches which are then illustrated in figure 1, before considering their combination.

2.1 The Engineering Approach

The engineering approach seeks to develop a series of methods so that the resulting construction is as useful as possible when the construction is finished. For example the processes by which a steel girder is made is such that, probably, it will have certain physical characteristics when made (torsion strength etc.). This approach focuses on what can be done *before* the system has been constructed, thus it concentrates upon developing methodologies and practices to do this. These methods may be based to some extent upon an underlying theory of systems and system construction, but on the whole they are systemisations of what has been found to work in the past. Thus essentially one: relates what one wants to methods that have been found in the past to produce this; makes a plan; and then implements it.

2.2 The Adaptive Approach

The adaptive approach takes an existing system and seeks to interact with the system, going through a cycle of testing its current properties and changing it, until it is acceptable. For example, one may train a dog so that it acquires the behaviours and habits that you need to guard your house (barking at strangers etc.). As with the engineering approach, this may be based upon some theory of the system or it may just be a matter of trial and error. This approach focuses on what can be done with a system *after* it is constructed and is achieved by comparing current properties against the desired properties and deciding what changes might move it from having the former to achieving the later.

2.3 Combining the Two Approaches

Of course, the two approaches are usually used together, and this is shown in figure 1 below. Thus however carefully a steel girder is constructed using established methods, it is tested for flaws before being used. Similarly one often has to make an initial system in order to be able to start adapting it and one often employs the engineering approach when one wants to structurally adapt parts of an existing system. Furthermore these approaches are often combined at different levels: engineering a bridge uses basic design forms which have been developed by a process of adaptation; and adapting the design of a car uses pre-engineered parts.

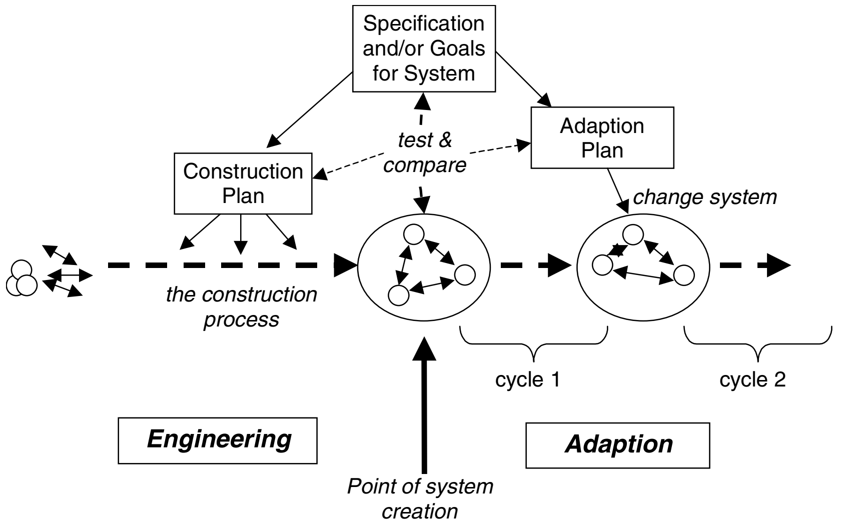


Fig. 1. An illustration of the engineering and adaptation phases before and after system creation

In the production of software systems, one typically first applies the engineering approach and then follow this with the adaptation approach – "10% implementation and 90% debugging", as the adage goes. However, this is not always the case for sometimes these occur in different combinations. For example, one might be faced with a legacy system, in which case one might be limited to the adaptation approach plus engineering additional wrappers, interfaces etc. If the adaptation process fails to get the system up-to-scratch one might be forced to re-engineer substantial sections of the system. Also these approaches might be used at different levels: thus one might *engineer* a mechanism to *adapt* some software; or *train* a human to *engineer* a compiler; or *construct* code in a high-level language to adapt lower-level code etc.

2.4 Using the Approaches Separately

Despite the fact that these two approaches are most effectively used together, there are large sections of computer science dedicated to eliminating the need for one or other of them. Thus genetic programming and other techniques in Machine Learning minimises the engineering phases at the object level, starting with randomised systems and adapting them from there. Similarly the formal methods community seems to wish to eliminate the adaptation phase and reduce system production to purely the engineering phase – attempt to make the production of software more akin to maths or logic. This unfortunate trend has been exacerbated by the split between the AI and ML communities with their different conferences, journals, approaches, traditions etc.

3 The Insufficiency of Engineering for SOSS

Elsewhere [8], Joanna Bryson and I criticise an over-reliance on formal design methods, where the engineering approach is focused on to the exclusion of adaptation. There I show a number of formal results, which are basically simple corollaries of Gödel [10] and Turing [20]. These can be summarised as follows: for a huge range of specification languages (e.g. those that essentially include arithmetic):

1. *There is no general systematic or effective method that can generate or find a program to meet a given specification.*
2. *There is no general systematic or effective method that, given a formal specification and a program, can check whether the program meets that specification.*

Where "general systematic of effective method" means one that could be implemented with a Turing Machine. These results hold for the overwhelming majority of classes of systems, including all those which include integer arithmetic (i.e. the vast majority of MAS). This illustrates the 'gap' between formal specifications and programs – a gap that will not be bridged by automation.

To illustrate how simple such systems can be, I defined a particular class of particularly simple MAS, called GASP systems (Giving Agent System with Plans). These are defined as follows. There are **n** agents, labelled: **1, 2, 3**, etc., each of which has an integer store which can change and a finite number of plans (which do not change). Each time interval the store of each agent is incremented by one. Each plan is composed of: a (possibly empty) sequence of 'give instructions' and finishes with a single 'test instruction'. Each 'give instruction', $\mathbf{G_a}$, has the effect of giving 1 unit to agent **a** (if the store is non-zero). The 'test instruction' is of the form $\mathbf{JZ_{a,p,q}}$, which has the effect of jumping (i.e. designating the plan that will be executed next time period) to plan **p** if the store of agent **a** is zero and plan **q** otherwise. Thus 'all' that happens in this class of GASP systems is the giving of tokens with value 1 and the testing of other agents' stores to see if they are zero to determine the next plan. This is illustrated in figure 2.

However GASP systems have the same power as Turing machines, and hence can perform *any* formal computation at all (a proof outline of this can be found in [8]). Since GASP systems are this powerful, many questions about them are not amenable to *any* systematic decision procedure. In particular, the above two results hold. Thus formal design methods can not provide a complete solution for system construction, and may only be effective for relatively simple systems (which accords with experience).

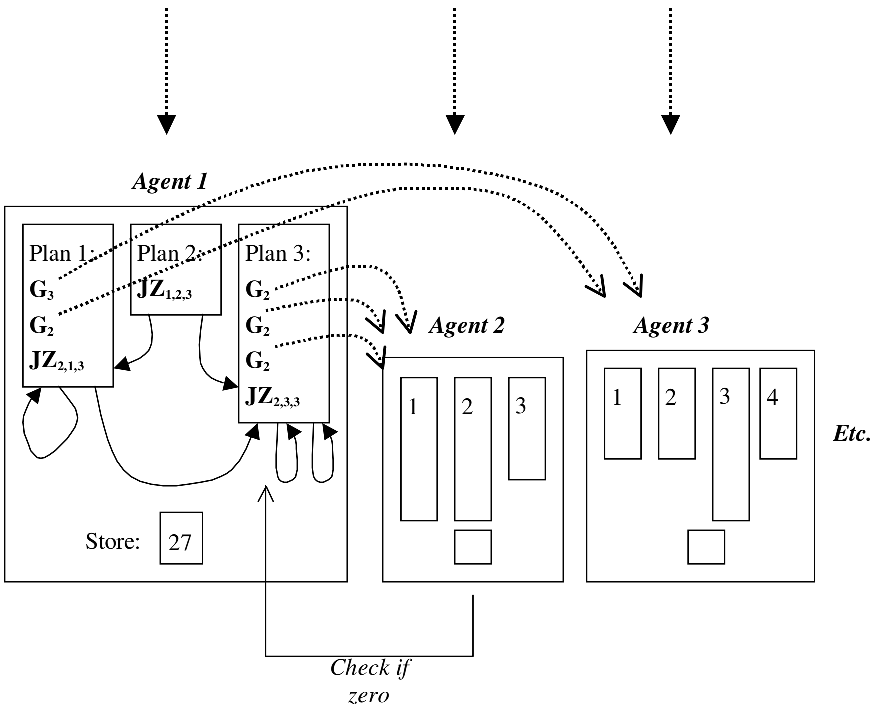


Fig. 2. An illustration of the working of GASP MAS: Each agent has a single store and a fixed number of very simple plans composed of a list of "give one" instructions and a final one of "if agent ***x***'s store is zero the go to plan ***a*** next, else plan ***b***"

Part of the problem seems to be the illusion that computational systems are predictable, simply because at the micro-level, each step of a computation is predictable. However, as the example of GASP systems shows, this is not the case. For even though working out what may happen next at any given stage is simple, it is impossible to compute many general aspects of their behaviour, from whether two machines will have the same effect in terms of their stores to whether a given machine will ever stop [5]. Thus we must give up the over-ambitious aim of complete reliance on the engineering approach when we consider MAS of even minimal complexity, and certainly for self-organised systems.

4 Producing Self-organised Software Systems (SOSS)

Since we can not totally rely on *engineering* self-organised MAS we need to consider also using adaptation as a *principle* method of useful system production, and not just as an after-thought to "fine tune" and "debug" systems we have already engineered. To illustrate this possibility I have evolved GASP systems to perform some simple tasks. These use a simple and untuned evolutionary algorithm with small populations of simple GASP systems over relatively short time runs, but nonetheless develop the desired properties. Of course, people have been evolving computational systems for about 40 years. The purpose of this section is to show: (1) that this can be done in very simple but effective ways with systems that are Turing-complete[2]; and (2) that this can be done with a MAS.

The evolutionary algorithm was extremely simple. A population of GASP systems were evolved. Each generation 1/3rd of the GASPs with the best fitness were preserved unchanged, the 1/3rd with worst fitness were culled, and the best 2/3rds mutated (with a 10% chance of any number in any plan being replaced by a new random number of the appropriate range) and entered into the population. This is called "Evolutionary Programming" [9] it can be seen as sort-of stochastic hill-climbing algorithm on a population. The algorithm is illustrated in figure 3.

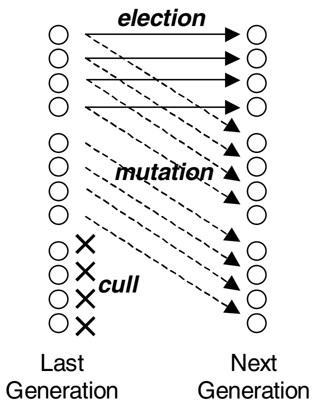


Fig. 3. The simple evolutionary algorithm applied to evolve GASPs: each generation the GASPs are ranked; the top 1/3 elected; the top 2/3 mutated; and the bottom 1/3 culled

This does not produce "open-ended" evolution, as can occur in Genetic Programming [13, 14], since the length of plans, the number of plans and agents is fixed. This could be fixed by including an operator to possibly increase these – this would probably result in the discovery of more sophisticated solutions [17].

4.1 Task 1: Long Periodic Pattern Development

To show that GASP systems producing outputs of increasing complexity can be evolved, I defined the fitness function as the period that the GASP system settled down into (if it did, the maximum otherwise) in terms of changes in the agents' stores. Thus each generation I ran each of 24 GASP systems for 500 time periods and at the end determined the period of repetition of the system. That is how far back one has to go to reach the same pattern as the last one. If there was no evidence of any such pattern (i.e. if the GASP does settle down to any repetitive behaviour so the time of the onset of this behaviour + the period of the repetition is > 500) it was accorded the maximum fitness and the evolution was halted.

[2] An interesting approach to evolving Turing Complete machines is [18, 19].

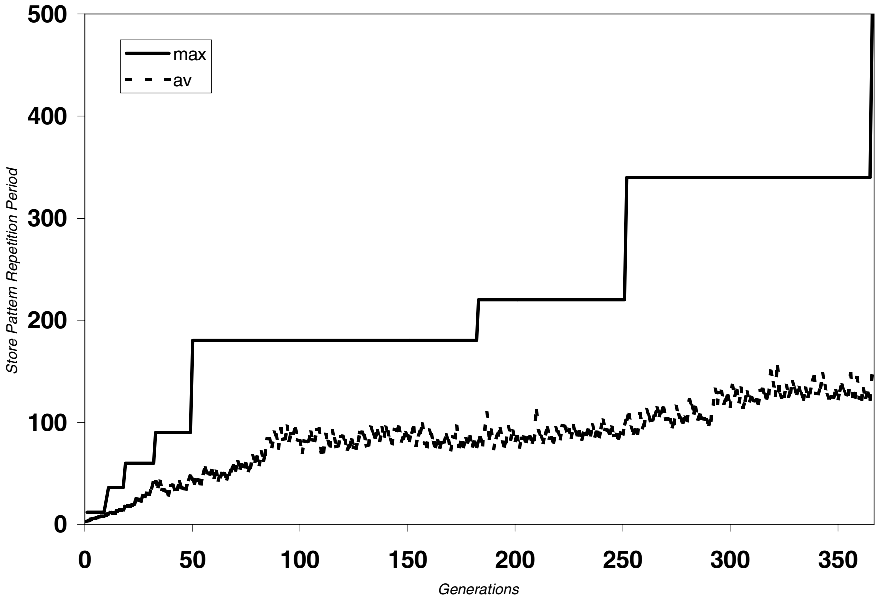


Fig. 4. The evolution of a GASP with a resulting repetitive period of over 500 time periods in 366 generations, with a population of 24 GASPs, each with 10 agents, each with 5 plans, each of which have ‘give lists’ of up to 3 instructions long (plus a “test for next” instruction)

The ease with which a GASP may be evolved to exhibit long periodic behaviour is strongly related to the number of agents and plans. A similar population of 24 GASPs of 10 agents, each with 10 plans achieved a periodic behaviour of greater than 1000 iterations in only 24 generations. In similar experiments I was able to evolve GASPs with periodic behaviour with high prime factors (there is an example in the appendix).

All that this shows is that it is feasible to evolve GASPs of increasing complexity. The next task is more difficult and more suited to the distributed nature of GASPs.

4.2 Task 2: Anti-avalanche Defence

The next task chosen is better suited to the nature of GASP systems, that is the distribution of their stores. The task here is to distribute its stores among its agents so that half of them have stores that are greater than those accumulated by a set of resevoirs to which shifting avalanches contributed to. One can think of the agents piling up the defences to keep out increasing piles of snow resulting from the avalanches. These avalanches are generated by a self-organised critical system and is known to produce avalanches whose distribution follows a power-law, and which is very difficult to predict [1]. The task of the GASP is to redistribute the units that are fed evenly (one to each agent) to the correct places to counteract the accumulating results of the avalanches. This is a continual race – the GASP is evaluated over its success at maintaining this over 25 cycles, but each time there may be a different pattern of inputs to the avalanche and a different pattern of avalanches. The overall set-up is illustrated in figure 4.

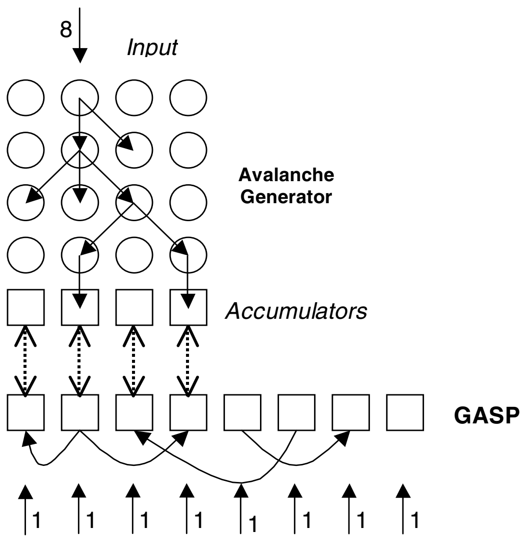


Fig. 5. An illustration of the target problem – the job of the agents in the GASP is to have more in their store that the corresponding accumulators receiving the results of the avalanches

The avalanche generator is a version of the basic 'sand-pile model' investigated by Per Bak and others [1]. It comprises of a set of piles, such that when a pile gets above a critical height it topples over onto adjoining piles, possibly causing them to topple etc. Units are constantly added, in this case to a pile along the 'top' edge. In this version when piles topple the units 'fall' randomly onto the three piles in the next row down in the adjoining columns (as illustrated in figure 4). The result is that the avalanche generator outputs, on average, the same number of units as was input but in irregular avalanches of various sizes. This makes it a difficult task to learn because the best GASPs in the long term will be those that ignore the particularities that give selective advantage in a single generation, but rather learns a more general strategy.

To show the advantage of the adaptive approach (and also to make the problem even more difficult) I set up the evolution so that the problem it is trying to solve changes during the evolution. The two versions of the problem are the 'variable input' and the 'fixed input' problem. In the variable input problem the input to the avalanche generator remains at a certain column position for a random number of iterations (in the range [1,10]) and then relocates to another randomly chosen position. This means that the avalanches will result with more being accumulated in the columns adjoining to the input position wherever it is, so in the variable problem this will change every now and then. In the fixed input problem the input is always at the first column, so there will be more long-term bias to the same output accumulators. Thus the variable input problem is more difficult to solve.

The GASPs were evolved against the variable input problem for the first 100 generations, then against the fixed input problem for 100 generations and back again to the variable input problem for the last 100 generations. In each generation the GASP is evaluated against 30 iterations of the GASP and avalanche generator. Each generation the avalanche generator is differently initialised with piles of random height below the critical height so the exact avalanche patters will be different every time – thus this is far from a static problem! Figure 6 show the success of this evolution over 31 runs.

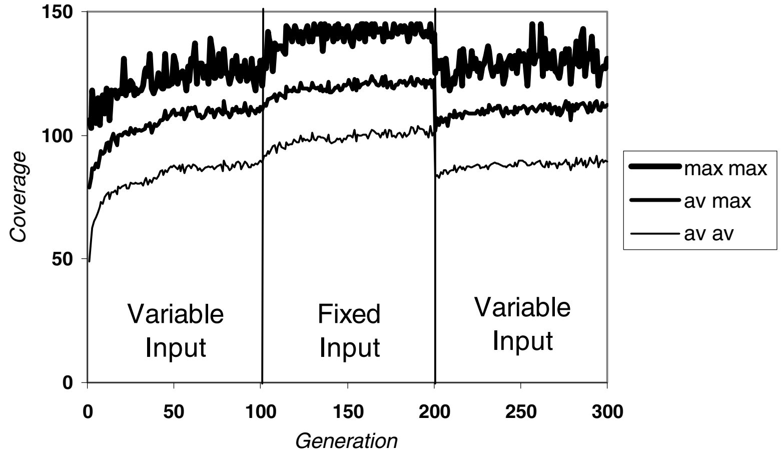


Fig. 6. Statistics showing the extent that the evolved GASPs covered the incoming avalanches (max=150). These are over 31 runs of the evolutionary algorithm, each with a population of 12 GASPs where each GASP is evaluated over 30 iterations. Bottom line shows the average over the 31 runs of the average coverages, next line the average of the maximum coverages and the top line the maximum of maximum coverages

As you can see, the GASPs evolve over the first 100 generations until they have learned to cover the avalanches to a certain extent. Then when the problem unexpectantly changes at generation 100 and becomes easier they quickly adapt to this. Finally when the problem is switched back to the variable input problem at generation 200 they have to relearn to cope with this (although this is much quicker than for the first time). This illustrates how an adaptive system (involving continual evolution) may be able cope with the unexpected better than an 'one-off' solution (however constructed). Simply taking the current best GASP is a crude way of using the learning achieved by the whole system, there are better ways (e.g. [17]).

One can imagine this style of system being applied to combat fraud where the type of fraud is being continually innovated. Beating a system that continually evolves is much more difficult than beating a static target. If the fraudsters (or virus writers!) invent systems to continually evolve their agents this might be the only effective defence. This is being investigated in the sub-field of artificial immune systems [3].

5 Putting the Production of SOSS onto a Sound Basis

If I am right that many SOSS will be evolved to a considerable extent rather than just designed, and that formal methods will not be able to ensure that such systems meet their specification, then we are left with a problem.

This problem is: *how are we to ensure that the systems we produce will perform satisfactorily when they are deployed in their operating context*?

The answer I suggest is this: *by systematically applying the classic experimental method used in the natural sciences*.

In other words, that we should make *explicit testable hypotheses* about the important characteristics of the systems we produce (by whichever means) and test these *experimentally* to determine: (1) their reliability and (2) their scope (i.e. the conditions under which they hold. These hypotheses should accompany the systems' publication, and be used by those who are considering using that system.

In addition to the hypotheses there should be sets of conditions under which it has been tested. Thus if a system has been run repeatedly under certain conditions (e.g. certain settings and parameter ranges) and it was found that in these circumstances the hypotheses held, them these circumstances should be appended to the hypotheses. As the system is tested in more circumstances this set should grow. When someone who wants to use the system for the properties listed in the hypotheses they should check that the circumstances it will be deployed under are covered by one of those that are listed as having been tested. If they are not, the person has the choice of either testing it themselves (and adding to the list if successful) or choosing another system. In this way there will be a slow co-evolution of the code, the hypotheses and the list of conditions as a result of the interaction of those using the system.

One can imagine an extension to the open-source method of code development so as to include some sort of open-access database for such systems, hypotheses and conditions of application. Programmers (or system growers!) would place their systems in the repository along with the normal documentation and some hypotheses and tested conditions of application. Others would test it under new conditions as they needed to and add this information to the database. Useful systems that were found to be reliable under sufficiently wide conditions would get to be used and tested a lot – systems whose scope was found to be narrow would be passed over. Eventually new variations of these systems would be created and the process continue.

The information might be as follows for each system (or major variation):

- $\boldsymbol{S}$, a description and/or method of production is described in sufficient detail to enable it to be made (at least with high probability). Each such $\boldsymbol{S}$ might have attached a sequence of:
    - $\boldsymbol{H}_i$, the hypotheses about $\boldsymbol{S}$ that encode the useful properties. Each $\boldsymbol{H}_i$ may have a list of $\{\boldsymbol{C}_{i,j}, \boldsymbol{S}_{i,j}\}$, constituted thus:
        - $\boldsymbol{C}_{i,j}$, the conditions under which each of $\boldsymbol{H}_i$ has been found to hold;
        - $\boldsymbol{S}_{i,j}$ is additional information accompanying each of the $\boldsymbol{C}_{i,j}$, for example: frequency of significance statistics concerning the occurrence of $\boldsymbol{H}_i$ under $\boldsymbol{C}_{i,j}$.

For SOSS that turn out to be useful, the $\boldsymbol{H}_i$, and $\{\boldsymbol{C}_{i,j}, \boldsymbol{S}_{i,j}\}$ need to be added and the continually refined in the light of experience. This makes the particular system, $\boldsymbol{S}$, much more useful to a potential user. Thus a 'meta-evolutionary' process will take place with the useful systems becoming selected and tested, and the unreliable and brittle systems being passed over.

It should be now clear how this is simply an application of the 'classic' scientific experimental method. In particular how it is directly analogous to the scientific process as conceptualised by Popper [16]. The world of software systems is one about which hypotheses are made, tested and developed. The crucial test of a system is not its relation (if any) to a designer's intentions for it but its proven performance in terms of the hypotheses about it. This marks a shift of emphasis away from verification to validation.

Now, of course, the method of construction and/or the process of adaptation are good sources for these hypotheses about system behaviour, but they are neither necessary (the only sources) nor sufficient (they can't be relied upon to be correct). Other hypotheses might come about solely from observing their behaviour (and maybe internal workings). Some others might be special cases of more general hypotheses concerning identified *classes* of system. A broad and important source for such hypotheses originate from other fields such as biology (e.g. Evolutionary Computation) or sociology (e.g. reputation-based mechanisms [4]).

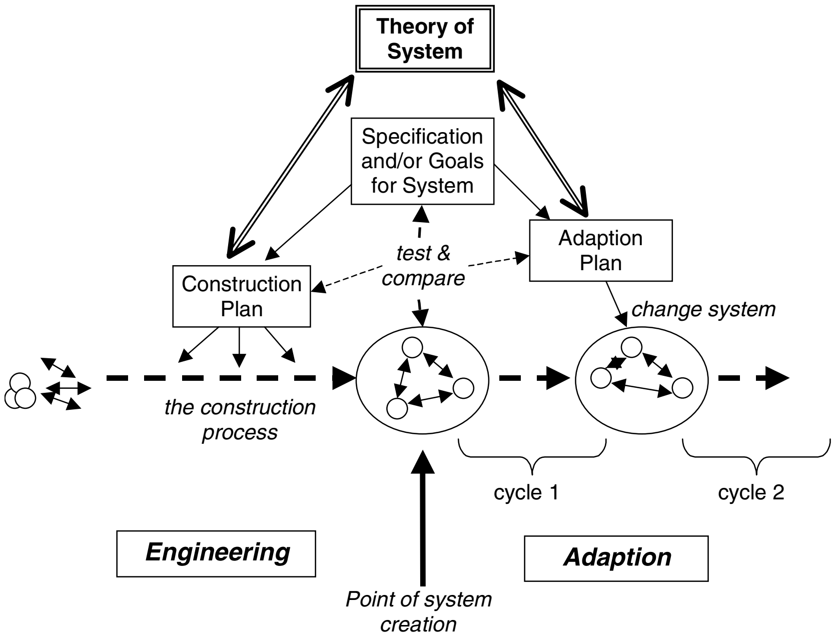


Fig. 7. An illustration of the relation of theory to the engineering and adaptation approaches

Thus there is a loose relation between: the plan of construction; the theory about the system; and any adaptation plan. For example: the system theory may be suggested by the construction plan; the adaptation plan may be informed by the system theory; the success of an adaptation plan may suggest a system theory; or a construction plan may be informed by the system theory. This set of relations is shown in figure 7.

Of course, as in science, *once* a theory has become established (by being extensively and independently tested), it can then be used to *deduce* things about the systems concerned. Formal deduction has a role with respect to *whole* complex and self-organised systems, but one that comes into its own only *after* a system theory has been experimentally established.

6 Example Cases

6.1 Hypothesising About Systems in Evolutionary Computation

There are areas of computing where something like an experimental method is widely applied, e.g. the field of Evolutionary Computation (EC). For example [15] proposes several hypotheses about the causes of bloat in GP populations and then tests them experimentally. This is indicative of the field. Whilst there are a few formal results and models (mostly of fairly simple cases and systems), the majority of the work could be described as experimental. Furthermore, in the sense that types of system are produced whose properties are broadly know and which are successfully applied in other systems and combined with other systems, it is successful.

However, more generally the hypothesising in evolutionary computation is usually: (1) specific to performance on a particular set of problems and (2) does not include the scope under which the hypotheses are found to hold. This makes it very difficult for a person considering applying such a system to come to a judgement upon its use for a different but similar problem. The hypotheses about the system are specific to particular problems, so one has to guess whether it is likely to be applicable to the new problem; and you do not know whether the system performance will extend to a new scope. Thus the reuse of such systems requires much individual experimentation, which deters many potential users.

6.2 Hypothesising About Tag-Based SOSS

'Tags' are features that are initially arbitrary but identifiable features of an agent – they have no effect upon the individual abilities of that agent but are observable to other agents. They can act as a (fallible) indication of cooperative group membership, when part of a suitably evolutionary process. They allow a dynamic but persistent maintenance of cooperation across a whole population even when defection is possible, without complex mechanisms such as: contracts, reputation or kin-recognition. This can occur because cooperative groups with similar tags are continually forming and persisting for a period before being invaded by a defector (which quickly destroys the group). Tag systems, and their possible relevance to SOSS are discussed in [12].

In common with many SOSS, tag-based systems are stochastic and fallible. That is, there is always a probability that cooperative groups will not occur. Thus one could never prove from its specification that the system would work as intended. However this effect seems robust over a range of settings and implementation variations. Thus its seems a viable hypothesis that such systems will result in significant amounts of cooperation over a reasonably wide range of settings.

David Hales has been working on such tag-based systems, work in which I have played a small part. As a result of inspecting the results of such systems, several hypotheses about the working of such systems (and hence the conditions under which cooperative groups might occur) have suggested themselves. One such condition that has been recently identified [11] is that the rate of tag mutation must be greater than the rate of defection in (or into) a cooperative group. This seems to be because it allows for new cooperative groups to form sufficiently often that there is always a significant 'population' of pure cooperative groups before the defection occurs in them. Thus although each group will inevitably be overrun with defectors, there are always enough cooperative groups in the total population to maintain the overall levels of cooperation. Thus, we not only have a hypothesis about a class of systems which has been observed, some of the conditions under which it is thought to occur, but also a mechanism by which it is though to occur.

The information published under the suggestion described in section 15 might be:

- $\boldsymbol{S}$ may be a description of one of the tag-based systems described as in [11];
- $\boldsymbol{H_1}$ might be that "the percentage of co-operators in the overall population is at least 30%";
- $\boldsymbol{C_{1,1}}$ might be the conditions under which each of $\boldsymbol{H_1}$ has been found to hold, for example: "the mutation probability of the tag > mutation probability of defection";
- $\boldsymbol{S_{1,1}}$ might be the frequency statistics concerning the occurrence of $\boldsymbol{H_1}$ under $\boldsymbol{C_{1,1}}$, in this case that "this occurred in 50 out of 50 trial runs over 30,000 generations".

7 Conclusion

'Engineering' and 'self-organisation' do not sit well with each other. The extent to which a system is engineered will constrain (as well as enable) what kind of self-organisation can occur. Likewise the extent to which self-organisation occurs will limit the scope for engineering since outcomes will be correspondingly undeducable. In other words, self-organisation will result in some outcomes that are not, *and can not be*, foreseen by any designer. An unwelcome surprise is always possible with self-organised systems. These surprises will often have to be dealt with by adapting the system after its creation. If we are to do better than trial and error in such adaptation we will need to develop *explicit hypotheses* about our systems and these can only become something we can rely on, via *replicated experiment*. This paper can be seen as a tentative step towards such an experimental method.

Acknowledgements

Thanks to the participants of the ABSS SIG of AgentLink II for their comments about the talk that eventually grew into this paper, especially Joanna Bryson. Thanks also to Scott Moss and David Hales for discussions on these issues (and everything else).

References

1. Bak, P. (1997) *How Nature Works: The Science of Self Organized Criticality*. Oxford, Oxford University Press.
2. Baram, Y., El Yaniv, R. & Luz, K. (2004). Online Choice of Active Learning Algorithms. *Journal of Machine Learning Research*, **5**:255-291. http://www.jmlr.org/papers/v5/baram04a.html
3. de Castro, L. N. & Timmis, J. I. (2002), *Artificial Immune Systems: A New Computational Intelligence Approach*, Springer-Verlag, London, September, 357 p.
4. Conte, R. and Paolucci, M. (2002) *Reputation in Artificial Societies – Social beliefs for social order*. Kluwer.
5. Cutland N.J. (1990). *Computability*. Cambridge: Cambridge University Press.
6. Edmonds, B. (2002) Simplicity is Not Truth-Indicative. CPM Report 02-99, MMU, 2002. (http://cfpm.org/cpmrep99.html).
7. Edmonds, B. and Moss, S. (2004) From KISS to KIDS – an 'anti-simplistic' modelling approach. Workshop on Multi-Agent Simulation and Multi-Agent Based Simulation (MAMABS), at AAMAS, New York, July. To be published in LNAI. (http://cfpm.org/cpmrep132.html)
8. Edmonds, B. & Bryson, J. (2004) The Insufficiency of Formal Design Methods - the necessity of an experimental approach for the understanding and control of complex MAS. In Jennings, N. R. et al. (eds.) *Proceedings of the 3rd International Joint Conference on Autonomous Agents & Multi Agent Systems* (AAMAS'04), July 19-23, New York, ACM Press, 938-945.
9. Fogel, L. J., Owens, A. J. and Walsh, M. J. (1967). *Artificial Intelligence Through Simulated Evolution*. John Wiley & Sons.
10. Gödel, K. (1931) Uber formal unentscheidbare Sätze der Principia Mathematica und verwandter System I. *Monatschefte Math. Phys.* **38**:173-198.
11. Hales, D. (2004) Change Your Tags Fast! - a necessary condition for cooperation? Workshop on Multi-Agent Simulation and Multi-Agent Based Simulation (MAMABS), at AAMAS, New York, July. To be published in LNAI. (http://cfpm.org/~david/papers/mabs2004.pdf)
12. Hales, D. and Edmonds, B. (2003) Evolving Social Rationality for MAS using "Tags", In Rosenschein, J. S., et al. (eds.) *Proc. of the 2nd Int. Conference on Autonomous Agents and Multiagent Systems*, Melbourne, July 2003 (AAMAS03), ACM Press, 497-503.
13. Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, MIT Press..
14. Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Subprograms*. Cambridge: MA, MIT Press.
15. Langdon, W. B. Terry Soule, Riccardo Poli and James A. Foster (1999) The Evolution of Size and Shape. In Lee Spector, William B. Langdon, Una-May O'Reilly and Peter J. Angeline (eds.) *Advances in Genetic Programming*, Volume 3. MIT Press, 163-190.
16. Popper, K. R. (1969) *Conjectures and Refutations*, London : Routledge & Kegan Paul,
17. Stanley, K.O. and Miikkulainen, R. (2004) Competitive Coevolution through Evolutionary Complexification, *Journal of Artificial Intelligence Research*, **21**:63-100. http://www.jair.org/abstracts/stanley04a.html
18. Teller, A. (1994) The Evolution of Mental Models. In Kenneth E. Kinnear, Jr. (ed.) *Advances in Genetic Programming*. MIT Press, 199-220.

19. Teller, A. (1996) Evolving Programmers: The Co-evolution of Intelligent Recombination Operators. In Peter J. Angeline and Kenneth E. Kinnear, Jr. (eds.) *Advances in Genetic Programming* , Volume 2. MIT Press, 45-68.
20. Turing, A.. M. (1936) On computable numbers, with an application to the Entscheidungsproblem. *Proc. Lond. Math. Soc*. **42**:230-65; **43**:544-6.
21. Zambonelli, F., and H.V.D. Parunak (2002) Signs of a Revolution in Computer Science and Software Engineering. 3rd International Workshop on Engineering Societies in the Agents World, Madrid, Spain. http://www.ai.univie.ac.at/~paolo/conf/ESAW02/

Appendix – An Example GASP

An example of a simple GASP evolved to have a repetitative period of 89. Figure 8 shows the set-up of the GASP and figure 9 a graph illustrating the cycle.

```
1, 1, [], 5, 3, 2
1, 2, [], 4, 2, 1
1, 3, [], 5, 2, 4
1, 4, [1], 3, 2, 4
1, 5, [3 4], 2, 1, 5
2, 1, [], 5, 4, 3
2, 2, [], 4, 2, 4
2, 3, [3 6 6], 1, 5, 1
2, 4, [6 5 4], 2, 2, 3
2, 5, [6 3 3], 3, 3, 2
3, 1, [], 4, 3, 1
3, 2, [6], 5, 3, 4
3, 3, [3 4 2], 3, 3, 4
3, 4, [4 4 5], 1, 3, 5
3, 5, [3 6], 1, 2, 1
4, 1, [], 1, 3, 3
4, 2, [], 1, 5, 5
4, 3, [3 3], 3, 3, 5
4, 4, [2], 1, 3, 1
4, 5, [3 2], 5, 5, 4
5, 1, [3 2 2], 5, 3, 5
5, 2, [1 6], 2, 3, 1
5, 3, [3 1 5], 2, 2, 4
5, 4, [1 2 1], 5, 1, 4
5, 5, [4 4], 4, 4, 4
```

Fig. 8. The plans of the agents in this evolved GASP system (agent number, plan number, give list, test agent, then plan, else plan)

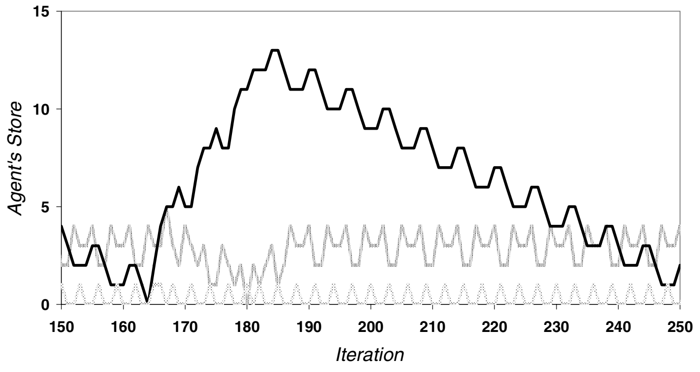


Fig. 9. A graph showing the 89-iteration cycle in 3 of the stores which results from this GASP executing

An Architecture for Self-organising Evolvable Virtual Machines

Mariusz Nowostawski, Martin Purvis, and Stephen Cranefield

Information Science Department,
University of Otago, Dunedin, New Zealand
{mnowostawski, mpurvis}@infoscience.otago.ac.nz

Abstract. Contemporary software systems are exposed to demanding, dynamic, and unpredictable environments where the traditional adaptability mechanisms may not be sufficient. To imitate and fully benefit from life-like adaptability in software systems that might come closer to the complexity levels of biological organisms, we seek a formal mathematical model of certain fundamental concepts such as: life, organism, evolvability and adaptation. In this work we concentrate on the concept of software evolvability. Our work proposes an evolutionary computation model, based on the theory of hypercycles and autopoiesis. The intrinsic properties of hypercycles allow them to evolve into higher levels of complexity, analogous to multi-level, or hierarchical evolutionary processes. We aim to obtain structures of self-maintaining ensembles, that are hierarchically organised, and our primary focus is on such open-ended hierarchically organised evolution.

1 Introduction

The rapid growth of complexity in different areas of technology stimulates research in the field of engineering of self-organising and adaptive computation systems. Adaptive software models refer to generic concepts such as *adaptability* and *evolution*. This, on the other hand, inherently leads to fundamental questions about the nature of open-ended uniform evolutionary processes: their essential properties, minimal requirements, architectures, models, and evolution of evolvability. Answers to some of these fundamental questions will lead to progress in automatic evolutionary design of computational machines and in engineering techniques for self-organising and self-adaptable software systems.

1.1 Traditional Methods

In common English usage *adaptation* means the act of changing something to make it suitable for a new purpose or situation. In software systems, the term adaptation is being used mostly, if not exclusively, with a second semantic meaning. What is usually meant by software adaptation is that the system will continue to fulfil its original purpose in new or changing circumstances, situations or environments. The adaptability in such software systems may be achieved by a feedback loops between the system, the controller monitoring and changing and adapting the system, and the environment itself. The system purpose is pre-defined in advance as a set of specifications, which are kept within the controller. The behaviour of the system is automatically altered if the expected outputs are outside of these pre-defined specifications. Such models operate analogously to the way automatic control systems work [15]. Most of them are based on top-down design and work well in limited environments, where changes in the environment can be predicted and constrained in advance [21]. Such adaptive systems are tuned to particular kinds and specific levels of change in the environment.

Most of the adaptability in traditional software systems is achieved via control mechanisms like in automatics. There is a central system, with a set of sensors and actuators, a controller, and an environment. Sensors sense an environment, the system and controller are tied via a set of feedback loops, and the controller tries to keep the system within pre-defined boundaries. This model can be implemented in a straightforward fashion, however it is static and must be applied in situations where it is possible to predict in advance all the changes and variations of the environment. To make things more robust and flexible, it is possible to implement into the controller an ability to learn, so the rules for changing the system become more dynamic, thereby enabling the entire ensemble to follow changes in more dynamic environments. Yet such systems still suffer from drawbacks associated with the simple control model. Even though the system shows some adaptability on another scale of complexity, there are limits of environmental change with which the system can cope. And these limits are pre-established with the structure of the learning mechanism itself.

1.2 New Requirements

Contemporary software systems, especially open multi-agent distributed systems, eg. [26], that may potentially be spread around the globe and interact with various changing web-services and web-technologies, are exposed to demanding, dynamic, and unpredictable environments where the traditional adaptability mechanisms may not be sufficient.

To imitate and fully benefit from life-like adaptability in software systems, that (at least in theory) might come closer to the complexity levels of biological organisms, we seek a formal mathematical model of certain fundamental concepts such as: life, organism, evolvability and adaptation. In this work we will concentrate on the concept of software evolvability.

The landmark step in understanding the evolutionary process of living organisms in natural life was done by Darwin [4], who proposed mechanisms by which purposeful adaptive changes take place via processes of random mutation and natural selection. Darwinian mechanisms postulate reproduction, the statistical character of change processes, and the process of elimination (after elimination the organism ceases to exist, i.e. is not alive anymore).

1.3 Computation and Biological Inspirations

In this work we use a theory of evolvable virtual machines, which exhibits adaptability and self-organisation. The model has been inspired by ideas that have been developed over the last decades. The roots of the proposed model can be traced back to the work of John von Neumann [38, 39], who submitted that a precise mathematical definition should be given to basic biological theories. This has been most prominently continued and extended by Gregory Chaitin [2, 3].

Some current research in evolutionary computation (EC) is emphasising information-centric methods that mirror Darwinian theory of random mutations and natural selection. This is visible in well-established computational optimisation methods, such as Genetic Algorithms (GA), Genetic Programming (GP), and their variations, such as assorted Artificial Life systems. Despite some successes, the typical simple single-layer evolutionary systems based on random mutation and selection have been shown to be insufficient (in principle) to produce an open-ended evolutionary process with potential multiple levels of genetic material translation, see e.g. [5, 41].

Our work proposes an alternative path, based on the theory of hypercycles [5] and autopoiesis [20]. The intrinsic properties of hypercycles allow them to evolve into higher levels of complexity, analogous to multi-level, or hierarchical evolutionary processes. We aim to obtain structures of self-maintaining ensembles, that are hierarchically organised, and our primary focus is on such open-ended hierarchically organised evolution.

2 Computational Evolution

2.1 Information-Centric Approach

It is believed by some that the information-centric approach is a correct, if not the only possible, path to pursue the research and make progress in the field of theoretical and computational biology [23, 3]. In our work, we use some of the basic concepts of the information-centric approach, and throughout this work we will use two basic notions of information as introduced by Shannon [33] and in Kolmogorov-Solomonoff-Chaitin algorithmic information theory [18]. We will refer to the Shannon notion as *information* and to the Kolmogorov-Solomonoff-Chaitin notion as *algorithmic information*.

There have been many more or less formal attempts to define life, complexity, organism, organism boundaries, and information content [32, 25, 19]. Some authors have attempted to give rigorous quantitative definitions of these concepts, in a formal deductive form [39, 3, 10]. Interestingly, authors coming independently from different sets of basic definitions and assumptions reached the same or very similar conclusions (e.g. [3] and [10]). According to theoretical and experimental work of most authors, the process of improvement in individuals and ensemble growth are best accomplished by carrying along all, or almost all, of the previously developed structures while new pieces of an ensemble structure are being added [34]. Simulations and statistical analysis in the fields of Artificial Life experimentally confirm the efficiency of this approach. Recent work in incremental reinforcement learning methods also advocate retention of learned structures (or learned information) (e.g. [30]). The sub-structures developed or acquired during the history of the program self-improvement process are kept in the program data-structures. It therefore comes as a bit of surprise that this general procedure is not being exhibited by any of (standard) evolutionary programming models [7] such as: Genetic Programming (GP) [16]) or Genetic Algorithms (GA) [40]. Although these evolutionary programming models are inspired by biological evolution, they do not share some significant aspects that are recognised in current evolutionary biology, neither can they be used (directly) in incremental self-improvement fashion.

We are seeking a new, robust, and flexible evolutionary model, that can accommodate meta-learning and incremental self-improvement, as well as hierarchically organised evolutionary processes.

2.2 Information Measure

Information is a measure based on a selection from a set of available choices. *Algorithmic information* is a uniform measurement of encoding information relative to the given computing machine (virtual machine).

The amount of information is based on the ability to make a correct selection from a given set. Let us consider a unique code $k \subset X \times Y$, i.e. $y = k(x)$, where $x \in X, y \in Y$, x represents a given condition, and y represents the correct selection. X and Y can be any sets, but in the context of finite state machines and discrete computation, one can treat them as sets of program blocks. Let us use index g to indicate a particular selection of x for a given y (goal). This model can be expressed now as $x_g = k^{-1}(y_g)$. Selection of a single unique condition x_g gives us all necessary information to obtain the output y_g: $I(x_g) = -log\, p(x_g)$, where $p(x_g)$ is the probability of picking a correct condition x, and I is the information content of a particular x_g [33, 1].

One of the possible ways to refer to the probability distribution $p(x)$ is to compare it with the reference distribution of the system. We can take as a reference a system that makes all possible choices with equal probability. Such a system would have maximum entropy (equivalent to the thermodynamical state of equilibrium). By *entropy* we mean an information theory measure which, when applied to an information source, determines the maximum channel capacity to transmit the source encoded according to a particular signal alphabet. The state of maximum information entropy we will refer to as a system in *abiotic equilibrium*. In such a state the probability of a correct selection of a given condition for a given output is uniformly distributed across all the possible conditions, and therefore all selections are equally probable. By calculating the difference between the actual $p(x)$ and this uniform abiotic distribution, one can calculate the information content needed to make a correct selection. (This is the difference between the given channel and one with maximum information capacity.)

In the context of incremental search methods through program search-space, as in the case of [31], $p(x)$ can be interpreted as the probability of executing a particular instruction during the course of program execution. The program itself can accumulate information about its environment and requirements by adjusting these probability distributions. A program can modify the probability distributions for different instructions at runtime. The difference between abiotic probability distributions (the initial uniform distribution) and the given probability distribution of a given system will be the measure of acquired information.

2.3 Darwinian Systems

Darwin's principle of natural selection is widely used in current computational models of evolutionary systems for optimisation or simulation purposes (in fact in Evolutionary Computation in general). Some authors regard natural selection as axiomatic, but this assumption is not necessary. Natural selection is simply a consequence of the properties of population dynamics subjected to specified external constraints. The main objective of the work of Darwin and Wallace [4] was to provide some basic insights into the process of evolution and the phylogenetic interrelations among species.

There are some inherent properties of conventional computational Darwinian systems which are sometimes overlooked. Darwinian systems rely on the concept of an environment with embedded self-replicating entities competing for resources and reproduction. For the model to be consistent, one has to postulate a stable species which competes for selective preferences and a stable reproduction of the best adapted species. In other words the model postulates that the selection operates purely on the individuals, hence there is only a flat single level of individuals which the evolutionary processes operate on. In such a model, there is a limit to the amount of the information content a stable species can have. Therefore the evolution of such a system is limited to a certain level of complexity defined by the threshold for maximum information content for a given setup/configuration of species. To overcome this threshold, further levels of selection and evolutionary information translation would need to be introduced into the system.

A second important aspect concerns random mutations. In real biological systems, due to overly complex and dynamic environments, the mutations can simply be a function of the environment. There is no need to postulate an external, god-like source of randomness. However, in computational models, the environments are highly regular and fixed, and the only way to introduce the necessary noise to the search process is by introducing an external source of randomness. This, as with certain random-search optimisation methods, can be useful, and in fact works quite well for some classes of problems (with the main computational techniques employed being Genetic Algorithms, Genetic Programming, and other evolutionary computation optimisation methods).

However in the context of adaptable, self-organising and open-ended evolutionary software system that exist in dynamic environments, it makes little sense to introduce additional external sources of randomness. By definition, the environment should supply all the randomness for the adaptable software system. If the environment stabilizes, the system should stabilize as well. It is a simple result of maximisation of the system aptness to the given environment. On the other hand, the learning mechanism involved in adapting the system to the given environment, or to the changes in the environment, may internally need to use some sort of probability distribution (that is represented internally, or externally via the environment itself). This is, however, a different matter to an artificially introduced external source of randomness, as in evolutionary computation.

One possible way of dealing with that is via bias-optimal search methods [17, 29], or via incremental search methods [31]. To narrow the search, one can combine several methods, for example it is possible to construct a generator of problem solver generators, and employ multiple meta-learning strategies. We will discuss some of the details further in the following sections.

3 Autopoietic Hypercycles

3.1 Hypercycle

Lets consider a sequence of reactions in which products with or without the help of additional reactants undergo further transformations. The *reaction cycle* or *cycle* is such a sequence of reactions in which some of the products are identical with the reactant of any previous step of the sequence. The most basic is a three-membered cycle, with a substrate, enzyme, and a product. The enzyme transforms a substrate into enzyme-substrate and then enzyme-product complexes, which in turn is transformed into a product and free enzyme. See Figure 1.

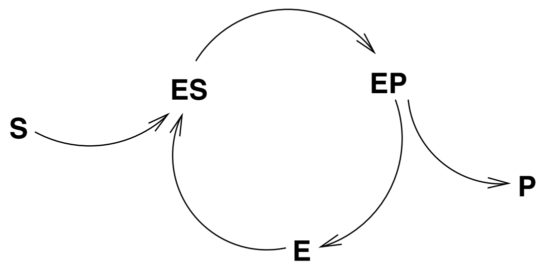


Fig. 1. An example of three-membered catalytic cycle: the free enzyme (E), the enzyme-substrate (ES) and the enzyme-product (EP) complexes all demonstrate a catalytic cyclic restoration of the intermediates in the turnover of the substrate (S) to the product (P)

The cycle as a whole works as a *catalyst*. Unidirectional cyclic restoration of the intermediates presumes a system far from energy equilibrium. This can be associated with a dissipation of energy into the environment. Equilibration occurring in a closed system would cause each individual step to be in balance: catalytic action in such a closed system would not be microscopically irreversible.

Lets now consider a reaction cycle in which at least one of the intermediates themselves is a catalyst (see work of Kaufmann on autocatalytic nets [14]). The simplest representative of this category is a single *autocatalyst* (or a self-replicative unit).

A system which connects autocatalytic or self-replicative units though a cyclic linkage is called a *hypercycle*. Compared with a simple autocatalyst or a self-replicative unit (which we can consider here to be a "flat" structure) a hypercycle is self-reproductive to a higher degree. This is because each of the intermediaries can itself be an autocatalytic cycle.

3.2 Autopoiesis

Following Maturana and Varela terminology [20], machines are unities which are made out of components. All components are characterized by certain properties. Machine components must operate according to certain relationships among their interactions and transformations that define the operation of the machine. The details of properties other than those participating in the interactions and transformations which constitute the machine are not relevant.

The *organisation* of the machine is defined as all the relations that define a machine as a unity and determine the dynamics of interactions and transformations which it may undergo as such a unity. The organisation of a machine does not specify the properties of the components which realise a concrete machine. The organisation of a machine is independent of the properties of its individual components, which can be any, and a given machine can be realised in many different manners by many different kinds of components.

The *structure* of the machine is defined as the actual relations which hold among the components which realise a concrete machine in a given space. A given machine (machine with fixed organisation) can be realised by many different structures. An organisation may remain constant by being static, by maintaining its components constant, or by maintaining certain relations between components constant which are otherwise in continuous flow or change.

An *autopoietic machine* is defined as a unity by a network of production, transformation, and destruction of components which: (i) through their interactions and transformations continuously regenerate and realise the network of relations that produced them, and (ii) constitute the machine as a concrete unity in the space by specifying the topological domain of its realisation as such a network. An autopoietic machine is an homeostatic or rather a relations-static system that has its own organisation as the fundamental variable which it maintains constant.

In contrast, a machine in which organisation is not autopoietic does not produce the components that constitute it. The products of such a machine are different from the machine itself. The physical unity of such a machine is determined by processes that do not enter into its organisation. Such a machine is called *allopoietic* [20]. Allopoietic machines have input and output relations as a characteristic of their organization: their output is the product of their operation, and their input is what they transform to produce this product. The phenomenology of an allopoietic machine is the phenomenology of its input-output relations. The realisation of allopoietic machines is determined by processes which do not enter into the organisation of the machine itself.

Self-organisation. An autopoietic system is considered to be a unity in the physical space. It is an entity topologically and operationally separable from the physical background. It is defined by an organisation that consists of a network of processes of production and transformation of components, molecular and otherwise, that through their interactions: a) recursively generate the same network of processes of production of components that generated them; and b) constitute the system as a physical unity by determining its boundaries in the physical space.

The important aspect of an autopoietic system is that it remains invariant in its organisation. The system itself can be deformed by external circumstances, but its internal organisation remains invariant. In other words, the self-organisation and self-maintenance of defining relations is inherent in the autopoietic model. Any change in the autopoietic organisation beyond a particular threshold is equivalent to the loss of identity, and the system disintegration.

Thus, an autopoietic system is defined as a unity by its autopoietic organisation, and all the transformations that it may undergo without losing its identity are transformations in which its organisation remains invariant. All autopoietic systems are therefore homeostatic. They maintain their own organisation constant through their operation. All the various unitary phenomena of an autopoietic system are constitutively subordinated to the maintenance of its autopoiesis.

Hierarchies. In conventional Darwinian systems all self-replicative units competing for selection are non-coupled. In other words, the selection forces operate purely on a single level: the level of individuals. This simply leads to a conservation of a limited amount of information, which cannot pass above a specified threshold. In hypercyclic systems, as distinct from conventional Darwinian systems, we deal with similar selective pressures. Note however, that in the hypercyclic case we also deal with integrating properties, and this allows for cooperation of otherwise competing units. Hypercycles are capable of establishing higher-order linkages. When inter-cyclic coupling is established, individual hypercycles may form hierarchies. In other words, the basic unit of selection may not be a single hypercycle, instead a whole chain of interrelated hypercycles. This is an important aspect of our work — exploiting the hypercyclic integrating properties and multi-level selective pressures.

If the autopoiesis of the component unities of a composite autopoietic system conforms to allopoietic roles that through the production of relations of constitution, specification, and order define an autopoietic space, the new system becomes in its own right an autopoietic unity of second order. The most stable condition for coupling appears if the unity organisation is precisely geared to maintain this organisation — that is if the unity becomes autopoietic. Therefore there is an ever present selective pressure for the constitution of higher order autopoietic systems from the coupling of lower order autopoietic unities.

In the theory of autopoiesis, unlike many other theoretical models of the process of life, the process of evolution is simply a side-effect, a consequence of limited resources, not the prerequisite. Whenever we deal with restricted resources, we have the selection and evolutionary pressures naturally occurring within our computational models. It is however important to recognize that life (or precisely: autopoietic systems) would still exist even if the process of evolution were not to occur in a system.

4 Evolvable Machine

4.1 Hierachical Computation

Some scholars believe that all sufficiently complicated systems are modelled best by hierarchical models [11, 27, 36]. In system sciences and cybernetics any system under investigation is thought of as a composition of multiple subsystems, each of which can itself be decomposed into subsystems, and this follows all the way down to a basic, fundamental level [34]. Hierarchies help us deal with complex phenomena by decomposing them into more manageable subsystems and investigating the interactions between these subsystems, one interaction at a time. The emphasis is placed on investigation of properties on different levels, mutual dependencies, and interactions between and within the hierarchy levels. Hierarchical decomposition of the problem space deals with complexity in a way that is natural and intuitive to humans.

4.2 Virtual Machines

Hierarchically organised virtual machines can be used as a specific computation model. Such a model is based on the traditional notion of computing machines, but extends it in certain aspects. The model discussed here provides a flexible and robust platform for experimentation with self-organisation and self-adaptability. It allows for a detailed analysis of different aspects of hierarchical complex system decomposition, together with the analysis of interactions between and within different hierarchical levels. This may help to understand a modelled problem or phenomenon better, giving us at the same time a robust and adaptable computing framework.

Formal Definitions. The following formalism is inspired by typical models of computing machines. More theoretical foundations for computing models from a programming perspective can be found in [24, 13].

From the Church-Turing Thesis we expect that all models of discrete computation, including the one presented here, will have the same properties as any other model of computation with respect to uncomputability and undecidability. This fact has some interesting and fascinating implications, see e.g. [6]. All the well known properties from computational complexity [18] are naturally exhibited by to the computational model presented here. This includes, for example: undecidability, the halting problem, and the concept of non-computable functions.

Definition 1. *A virtual machine or a computing machine (or just a machine for short) is a tuple $M = (K, \Sigma_{in}, \Sigma_{out}, \delta, s)$ where K is the set of states and $s \in K$ is the initial state. Σ_{in} and Σ_{out} are sets of input and output symbols, respectively, referred to as input and output alphabets. δ is a function that maps $K \times \Sigma_{in}$ to $K \times \Sigma_{out}$, and is called the program. We say δ (or the program) runs on machine M. Remember that formally δ is an integral part of the machine itself. The notation $M(x)$ represents the output of machine M given the input sequence x. $M(x, y)$ represents the output of machine M given the input sequence x followed by the input sequence y.*

Definition 2. *Suppose that f is a function from $(\Sigma_{in})*$ to $(\Sigma_{out})*$, and let M be a machine with input and output alphabets Σ_{in} and Σ_{out} respectively (the symbol $*$ has the usual meaning of "set of all possible sequence from a given alphabet"). We say that M computes f iff for any string $x \in (\Sigma_{in})*, M(x) = f(x)$. If such machine M exists, f is called a recursive function. We also say that function f is computed by machine M.*

Definition 3. *If for machine $M = (K, \Sigma_{in}, \Sigma_{out}, \delta, s)$ there exists a machine $M' = (K', \Sigma'_{in}, \Sigma'_{out}, \delta', s')$ which computes δ, we call machine M a recursive virtual machine or recursive machine for short. We call program δ' an interpreter of M, and we say an M interpreter runs on machine M'. We have $\forall x \in (\Sigma_{in})*$, $M(x) = M'(\delta, x)$.*

Definition 4. *Suppose we have a machine $M = (K, \Sigma_{in}, \Sigma_{out}, \delta, s)$ and there exists machine $M_c = (K_c, \Sigma_{in_c}, \Sigma_{out_c}, \delta_c, s_c)$, where $\Sigma_{in} \subseteq \Sigma_{in_c}$ and machine $M' = (K', \Sigma'_{in}, \Sigma'_{out}, \delta', s')$ where $\Sigma_{out_c} \subseteq \Sigma'_{in}$ and $\Sigma_{out} \subseteq \Sigma'_{out}$. If $\forall x \in (\Sigma_{in})*$, $M(x) = M'(M_c(x))$ then we say that δ_c is an M compiler, and we say M_c compiles M into M'.*

The emphasis in the conceptual framework presented above is to treat algorithms and running programs as *machines* (*recursive virtual machines* to be precise). This along with the notions of compilers and interpreters is discussed at length in [13]. The above definitions do not make any assumptions about the number of states a given machine can have, nor about the storage capability. All possible models of computations, and different computer/algorithm architectures fit the above definitions. For example one could use $\Sigma \subseteq Real$ to perform analog computation on real values. It can be shown that the proposed conceptual framework is a simple extension of the theoretical models of computation such as Turing machines and Universal Turing machines [12, 24].

Proposition 1. *Let machine* $M = (K, \Sigma_{in}, \Sigma_{out}, \delta, s)$*, with finite input and output alphabets* $\Sigma = \Sigma_{in} = \Sigma_{out}$*,* $\{\sqcup, \triangleright\} \in \Sigma$ *and* $\{h, y, n\} \in K$*. In other words the alphabet contains two special symbols, the blank and the first symbol, and there are three extra state symbols, namely:* h *the halting state,* y *the accepting state,* n *the rejecting state. We define three additional symbols, representing cursor directions:* $\leftarrow$ *for "left" and* $\rightarrow$ *for "right" and* $-$ *for "stay". If* δ *maps* $K \times \Sigma$ *to* $K' \times \Sigma$*, where* $K' = K \times \{\leftarrow, \rightarrow, -\}$ *then we say that machine M is a Turing machine.*

5 The Architecture

We can model artificial and naturally occurring phenomena as a chain of virtual machines. One possible perspective on artificial life or evolutionary systems is to focus on a tower of compilers and/or interpreters. The concepts of chaining and stacking compilers and interpreters is discussed in detail in [13]. The other approach is to use more traditional functional decomposition. All computing programs, including all evolutionary computation models can be represented as a chain of compilers and/or interpreters, with different functional partitioning on each level. The way this chain is constructed and how all its elements interact with each other is a principal concern of our hierarchical computing architectural approach.

5.1 Vertical and Horizontal Decomposition

Following the formal definitions, a machine can be statically represented as a program string, consisting of a prefix, together with some instructions following this prefix. The prefix itself can be decomposed into another prefix and another program, and so on. This is called *vertical decomposition*, or a *vertical hierarchy*. Another type of decomposition is based on dividing a given machine into interacting parts – this is called a *horizontal decomposition*. Formally, a vertical hierarchy is based on stacking interpreters and/or compilers [13], see Figure 1. A horizontal decomposition is based on splitting a single machine into two or more machines, see Figure 2.

Existing examples of vertical hierarchies are all sorts of (real-life) interpreters and compilers. For example given a Pascal interpreter written in Java we would have: program written in Pascal $\longrightarrow$ Pascal virtual machine (written in Java) $\longrightarrow$ Java virtual machine (written for example in C) $\longrightarrow$ C virtual machine $\longrightarrow$ etc., where the arrow reads as "runs on" as defined in Definition 3.

An example of horizontal partitioning would be a functional partitioning of a single individual virtual machine. Let us imagine that we have a machine that can compute two operations on the natural numbers domain: addition and multiplication. If we perform functional partitioning, we can end up with two virtual machines, each computing a single operation, multiplication or addition, respectively. The union of these two gives us the original single machine.

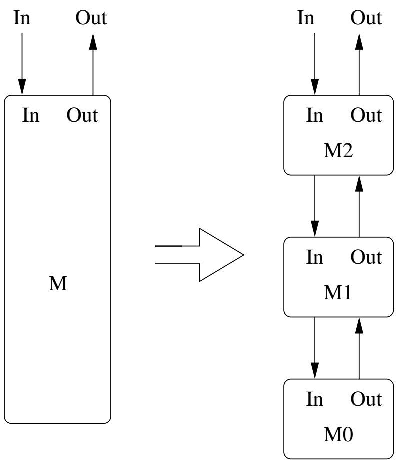


Fig. 2. A vertical split of machine M into a tower of machines M0, M1, M2

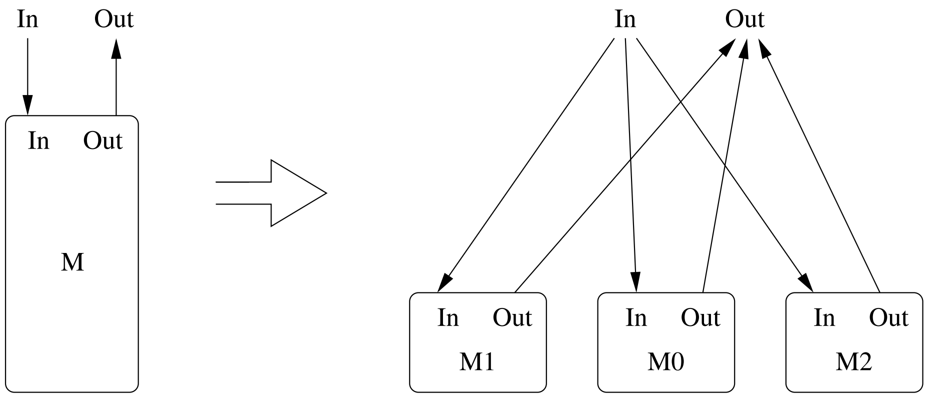


Fig. 3. A horizontal split of machine M into machines M0, M1, M2

One can enumerate through all the machine levels, starting from the base (fundamental) level M_0, up to the final highest-level machine, M_n. The actual input (instructions) are fed to the machine M_n. It is important to remember that, in fact, there is no special distinction between the *program* running on a virtual machine and the program emulating a particular machine itself.

All the interacting virtual machines are connected by their input/output streams. The hierarchical structure of that composition can have different forms, depending on the particular phenomena at hand. It can be a simple linear structure, or it can be a tree-like structure. In general it is a directed graph, with cycles, with self-referencing nodes, and possibly with complicated interdependencies (see Figure 5.1).

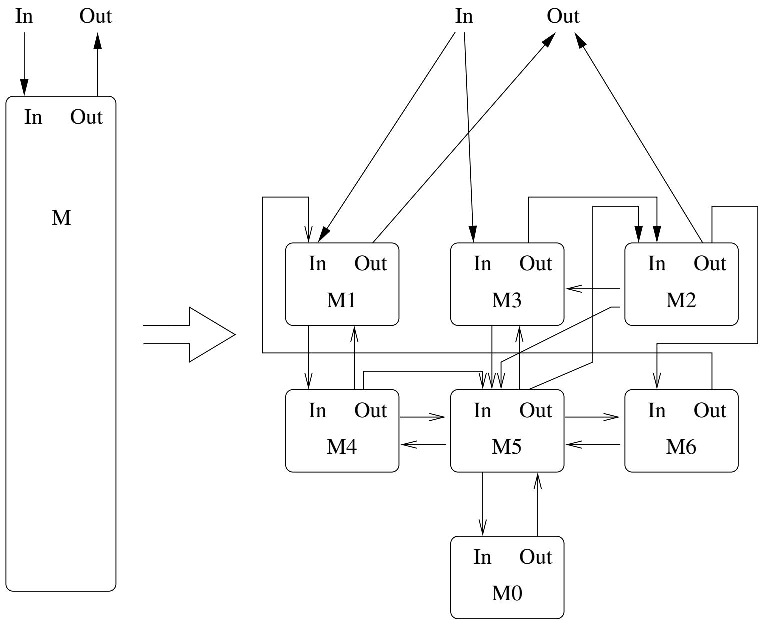


Fig. 4. The example of possible dependencies between machines after decomposition

5.2 Partial Equivalence

Some machines can be fully or partially equivalent to others; for example a Pascal virtual machine written in C and a second one written in Java are always perfect and fully equivalent Pascal virtual machines, even though they use completely different machines on the lower level. Note that even though these two Pascal virtual machines have different machines below them, they can have exactly the same virtual machine one level down, for example a virtual machine for a particular operating system.

One can have a partial Pascal virtual machine that accepts a subset of all possible programs generated in Pascal. This is referred to as *specialisation*. On the other hand it is also possible to have a Pascal virtual machine accepting a subset of expressions from the C language, in addition to normal Pascal programs. The process of adding features to the language and enhancing the input language for a given machine is called *conservative extension* [13].

Some machines can be recursively executed on themselves, for example a Java virtual machine interpreter written in Java and executed on a Java virtual machine interpreter. Some machines can be functionally equivalent even though they use completely different language syntaxes or alphabets, for example expressions in prefix, postfix or reverse Polish notations. All these properties are well known in computer science, in which specific languages, interpreters and compilers flourish.

Suppose the problem at hand is coded in such a way that the solution can be expressed as a string of symbols from some language, L. For some languages finding a solution string is easier than for others: coding the problem is the key issue in solving the problem. In a sense the language L captures and exploits some of the properties of the problem. This is one of the main features of the proposed approach. With recursive virtual machines we have the necessary framework to model the transformations of a problem representation from one language to another, and we are able to translate the original problem into a more easily solvable equivalent.

5.3 Decomposition Limits

A particular level from the hierarchy is treated as a virtual machine that provides some functionality to the other level immediately above it, and uses the level below to have the computation performed. In other words a particular machine accepts input from one level, uses other levels to perform computation, and then returns the results back to yet another adjacent level. The highest level of the chain of machines accepts some input (instructions), interacts with the level below it by sending/receiving some input/output, and returns some outputs (results) back. Similarly to the base level, what we consider the highest level is also arbitrary. There is always a virtual machine feeding the instructions and accepting the results (e.g. a computer program or a human operator).

A given machine in a chain is formally equivalent to an interpreter or compiler of another machine located above it. The first, the base level is the very first interpreter, which we assume as being executed on some universal virtual machine (UVM). In the case of digital computers (and for the sake of simplicity) we can without loss of generality assume that the base level machine is equivalent to the Universal Turing Machine [12]. Of course, this is an arbitrary choice, and the decomposition could be carried further, treating the UVM itself as a virtual machine, running on some software/hardware platform and so on, all the way down to electrical and/or chemical reactions and some physical processes[1].

6 EVM Implementation

6.1 Yet Another Language?

There exist many programming languages developed within the field of Evolutionary Computation. Many employ usual higher level programming languages designed for human programmers (such as Lisp for the original formulation of tree-based Genetic Programming [16]); some are developed with an evolutionary process in mind [35, 28], and others are developed for machine processing and recursive program manipulations [31]. Some of the languages are highly specialized, and provide the evolutionary mechanisms with a bias towards a particular solution subspace. However, none of these languages provides mechanisms to manipulate levels – a property needed for our EVM implementation. There are other features we want our base machine language to possess, that none of the existing languages have. For example, we want the language to be capable of redefining itself. That is, the primitive instruction set must allow the evolutionary process to restructure and redefine itself. Also, we want a programming language that is

[1] Actually, according to [9] we have no reason to stop there, and we can decompose the system further, based on the idea that physical phenomena itself are running on some (digital, in the case of Fredkin's theory) virtual machine.

highly expressive, that is, we want solution programs to typically encountered tasks to be short. And also, we believe that there are efficiency advantages for a language whose solution spaces are highly recursive.

A programming language used for search in Evolutionary Computation plays an important role – some programming languages are particularly suited for some, but not for all, problems. One of the appealing aspects of a multi-level search process is that, in principle, it can define a new base level and a completely new programming language that is specialized for the given task at hand. We want to exploit this property.

Some of the existing languages possess some of these desired properties, but no single one of them possesses all of them. This is why we have designed our own specialized programming language. The principal objective of the overall programming language is to facilitate searches for specialized languages for a given set of problems, and we want the EVM to facilitate that. Even though there is currently a concrete implementation of the base machine for the EVM system (the primitive instruction set), we treat it only as a temporary list. We are working on redesigning the base machine to better suit and to help with the search of program generators. So far, we have obtained some results suggesting the need of some of more computationally intensive primitive instructions to be included into the base machine. On the other hand, some of the existing instructions are rarely being used, and will be removed in the next iteration of our implementation.

6.2 Computing Model

The hierarchical computing model presented in the previous sections can be implemented in multiple ways and in many physical programming languages. It should be understood that the implementation presented below is only one of many possible implementations, and the choice for this particular implementation as distinct form other computing architectures, is somewhat arbitrary. On the other hand, we have paid considerable attention in order to make the implementation as flexible and robust as possible and to facilitate different configurations and different experiments in order to fine-tune the instruction set and the overall computing architecture for general-purpose use.

Our initial implementation of the EVM architecture is based on a stack-machine, such as Forth [22], or Java Virtual Machine (JVM) [37]. In fact, with small differences, it is exactly the same as an integer-based subset of a JVM.

The main architectural component, similar to the JVM, is the so called *execution frame*. The schematic view of the execution frame is presented on Figure 6.2.

The basic data unit for processing in our current implementation is a 32-bit signed integer. The basic input/output and argument-passing capabilities are provided by the operand Stack, called here *Data Stack*, or for short *Stack*. Data Stack is a normal integer stack, just as in a JVM for example. All the operands for all the instructions are passed via the Stack. The only exception is the instruction `push`, which takes its operand from the Program List itself. Unlike the JVM, our virtual machine does not provide any operations for creating and manipulating arrays. Instead, it provides instructions for and facilitates operations on lists. There is a special stack, called *L-stack* for storing integer-based lists. The L-stack is implemented as an a-stack (a-stack is a special way of implementing stack, such that its top element is stored as a special register/variable). The L-stack top element is stored in a special list called *List*. In other words, *List* contains the “actual”

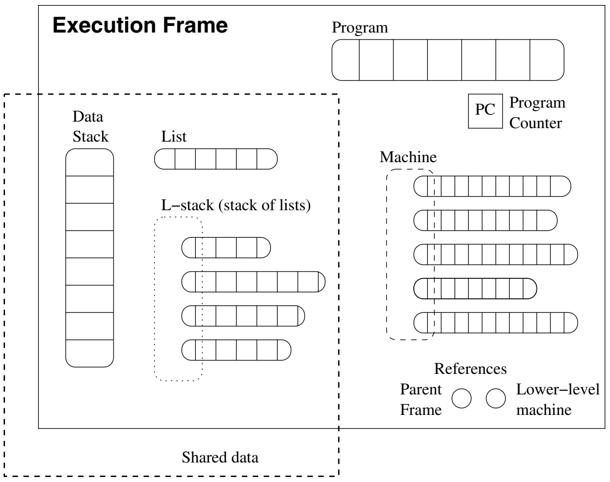


Fig. 5. Schema of the execution frame

top element of the L-stack, and the top element that is on the stack of the L-stack is the second topmost element, and so on. The decision to implement L-stack based on a-stack is purely for efficiency purposes. However it also makes all the instructions that operate on a List more natural and more intuitive, as they operate on an actual list, not on the top element of the L-stack. Both, Data Stack and L-stack (together with the List), are shared between multiple Execution Frames that share a common thread of execution.

There is a lower-level machine handle attached to each of the execution frames. This is a list of lists, where each individual list represents an implementation of a single instruction for the given machine. So, in other words, the machine is a list of lists of instructions, each of which implements a given machine instruction. Of course, if the given instruction is not one of the Base Machine units, the sequence must be executed on another lower-level machine. The Base Machine implements base instructions that are not reified further into more primitive units. These instructions will be discussed later in this section.

The program in the Execution Frame is represented as a list of integers. The program counter (PC) points to the current instruction in the program list. The PC is itself an integer, and that limits the theoretical length of any single program to 2^{31}. This is however lowered by the maximum length of any list in the system, which is currently set to 100000. Each instruction in the program list points to the appropriate instruction in the machine. If the value of a given instruction in the program is bigger than the number of instructions in the given machine, the index of the instruction is calculated as $index = instruction \ mod \ machine_{size}$, where $machine_{size}$ is the number of instructions in a given machine.

Each Execution Frame can reference the parent frame. The parent frame is responsible for creating and initializing the given frame. Base-level frames are those top-level frames that do not have any parent frame (the reference is $null$).

6.3 Execution Model

As described in the previous subsection, the EVM program is represented within the Execution Frame as a list of integers. Each integer (modulo Machine number-of-instructions) points to an individual machine instruction, that implements a particular behaviour of a given program instruction.

There are two possible situations. First, the instruction of a machine may be a primitive instruction, or *EVM operation*. In that case, the execution of a program instruction is simple: the behaviour just happens within the current Execution Frame. For example, if our program contains an integer, that points to `add` operation on the Base EVM Machine, this operation will take two arguments from the Stack, will add them together, and will put the result back on the Stack.

The second situation is when the instruction of the machine is a composite instruction, i.e. a list of instructions for lower-level machine. In that case, the execution of a program instruction proceeds as follows. First, a new Execution Frame is created. This new Execution Frame is initialized with the Stack of the parent Execution Frame (all Execution Frames in the same thread of execution share the same stack for parameter and return value passing). The PC of the new Execution Frame is set to `zero`, and the Program List is set to the Machine Instruction List. Then, the control is passed to this new Execution Frame, which executes this “subprogram”. Once done, the control is switched back to the parent Execution Frame.

The EVM is Turing equivalent, therefore there exist EVM programs that can run indefinitely. Each thread of execution has an instruction time limit, to constrains the time of each program in a multi-EVM environment. That is, each execution thread (single program) has a maximum number of primitive instructions that it can execute. Once the limit is reached, the program halts.

6.4 Instruction Set

As noted before, the instruction set is modelled after the Forth [22], and Java Virtual Machine [37] instruction sets.

The instruction set is divided into several categories, which we describe here briefly. The first category is “Stack and general operations”. This includes pushing constants onto the Stack, popping, swapping, rolling, duplicating, etc. Some of the example instructions are: `const1, const0, pop, swap, roll, dup`.

The second category is L-stack operations. There are operations for appending, removing, and manipulating lists on the L-stack. It also contains operations to transfer lists between L-stack and Stack. Some of the example instructions are: `lpop, lpopn, lswap, ldup, ldepth`.

The third category contains List operations, and includes transferring elements between List and Stack, and manipulating List elements. Some of the example instructions are `prepend, append, load, store, length, rmf, rml, rm, rmn`.

The fourth category contains operations for manipulating the Machine list of lists. This includes similar operations to L-stack operations, but here they refer to Machine. Some example instructions are: `mappend, mprepend, mload, mstore, mrmf, mrml, mrm, mins`.

The fifth category comprises the three level-related operations. These are `spawn`, `up` and `down`. We will discuss them in more detail below.

The sixth category contains all the control instructions. This list is based on the JVM control operations, and contains the following instructions: `ifeq, ifneq, iflt, ifle, ifgt, ifge, goto, jmp`. There are three extra instructions. `exec` takes the content of the List, and instantiate new Execution Frame and executes as if List is a program to be executed. This is equivalent of executing dynamically created subroutine. The other two instructions are two "search and jump" instruction. They take the element from the operand stack, and search forward (`jmpsf`), or backward (`jmsb`) to find same element in the program list and jump to the next instruction following that element.

The seventh and eighth categories contain all the Logic and Arithmetic operations. Logic operations are: `shl, shr, ushr, and, or, xor`, and `not`. Arithmetic operations are: `add, inc, sub, dec, mul, div, rem`, and `neg`.

6.5 Multi-level Computation

The key feature of the EVM, apart from its clean and elegant "zero operand architecture" [8], is that it offers multi-level processing. This is like having unrestricted reflection and reification mechanisms built-in for the virtual machine itself. The computing model is relatively fixed at the lowest-level, but it does provide the user with multiple computing architectures to choose from. The model allows the programs to reify the very virtual machine on the lowest level. For example programs are free to modify, add, and remove instructions from or to the lowest level virtual machine. Also, programs can construct higher-level machines and execute themselves on these newly created levels. Not only that – a running program can switch the context of the machine, to execute some commands on the lower-level, or on the higher-level machine. All together it provides unlimited flexibility and capabilities for reifying EVM execution.

Let us consider a particular example. Imagine, that we are tasked with writing a program to add a given number N of integers. All N integers are provided on the Data Stack, and the result is expected to be on the stack. We make it an incremental problem, by iterating from $1 \ldots N$. On the base machine we only have arithmetic operations, such as `add`. The `add` operation takes two arguments from the data stack, adds them together, and puts the result back on the top of the stack. The task of adding two numbers can be solved by a program with one instruction `add`. The task of adding three numbers requires two `add` instructions, and so on. If we generalise it for N, the simplest program would look like this:

```
add add add ... add   /* (nth-1 instruction) */
```

Given that the solution for N would be provided as a prefix for the program that must solve the task for $N+1$, the probability of randomly generating the postfix code for the $N+1$ problem would be $1/|BM|$, where $|BM|$ represents number of instructions in the base level machine.

With the multiple-levels, the prefix however can specialise a higher level machine for the "adding numbers" problem. This can be easily achieved by creating a higher level machine, with only one instruction, that adds two numbers. The program would look like that:

```
push add   /* pushes code for ADD instruction */
depth      /* pushes 1 on the stack */
popn_1     /* appends 1 element into the list*/
const_1    /* pushes 1 on the stack */
spawn      /* creates a higher level machine */
up         /* changes to the higher level */
0  ...  0  /* instruction repeated nth-1 times */
```

Note, that it takes only 7 instructions to construct and switch to the higher level machine, whose sole purpose is number addition. The higher level machine is a specialized machine that can only add numbers. It does not matter what the $N - 1$ instructions are that are actually appended to the end of the program. The program will always correctly add N numbers. In this case, the probability of solving the $N + 1$ problem becomes 1, as any of the added instructions would map to the single instruction on the higher level machine.

Of course, this is an extremely simple case, but it demonstrates the specialization capabilities of a multi-level computing machine. The governing idea is to create a custom specialized language and to solve the problem in that language, instead of trying to solve it in the original language of the base machine.

6.6 EVM and Its Expressiveness

We were inspired by the expressive power of a simple programming language designed by Schmidthuber [31]. However, we noted, that some of the recursive functional constructs he introduced in his language could be done more simply (i.e. making them shorter) in our own language for EVM.

For example, in Schmidhuber's language a recursive call to define and to calculate the factorial of N, assuming N is placed on top of the data stack, takes 14 instructions and has the following form:

```
c1 c1 def up c1 ex rt0 del up dec topf dof mul ret
```

In our language it is only 8 instructions, and the program looks like this:

```
dup halt0 dup dec lpush_p mappend mlcall mul
```

Note, our `dup` is equivalent to Schmidhuber's `ex`, `halt0` is equivalent to `rt0`, `mdepth` to `topf`, `mlcall` to `topf` and `dof` executed together. `lpush_p` copies the current program list into the List, and `mappend` appends the newly defined program from the List as a new primitive instruction to the base machine. `mlcall` executes the last instruction from the current machine instruction list. Our program is shorter, because (a) we do not need to define the number of arguments and result values, (b) we do not need to explicitly call `ret`, and (c) in our language defining a new program based on the current program in the List takes one instruction, and based on the program in the Program List, takes only two instructions.

Similarly, in the case of a context free grammar problem described by Schmidhuber [31], his solution is 5 instructions long, and looks like this:

```
defnp c1 calltp c2 endnp
```

Our code takes only 3 instructions:

```
c1 rwhile c2
```

`rwhile` is a “recursive while” instruction, that works in the following way: it checks the top of Data Stack; if there is value 0, it halts, otherwise, it forks to a recursive call back to the original program.

Because the EVM is more expressive, any search method, including Schmidhuber’s optimal problem solver, should find the appropriate solutions faster.

6.7 Program Ontogeny

Let us view a program input as a sequence of integers on the Data Stack, and consider a single program loaded into the Program List in the Execution Frame. Both, the input, and the program, can be divided into two subsequences indexed 1 and n, in such a way, that the first subsequence of a program, P_1 (program with index 1) reads all the data from the D_1 subsequence. P_1 will produce some results on the Data Stack, and it can also manipulate the program list itself. Hence, the remaining subsequence on the Data Stack is now longer, and the remaining program on the program list may differ from the original program list. If this process is repeated recursively, the final program, and the final data that this program reads, will be remapped from whatever was originally on the Data Stack, and inside the Program List. This process is referred to as Program Ontogeny. It demonstrates the development of the final (mature) stage of a program from some initial (larval) stage, through a sequence of transformation steps.

6.8 EVM and Hypercycles

In the current implementation of the EVM architecture, we employ two initial designs that facilitate the hypercyclic dependencies. One of them is based on the notion of self-replication of the EVM programs. The other is based on the notion of cyclic data flow. They each, in a way, complement each other. We will describe them below based on simple examples.

Self-replication. If a given program produces an output, and this output is identical to the program that produced it, we have a self-replicating EVM program. In other words, we have a program that can calculate (produce) itself. If a program P_1 produces another program P_2, such that P_2 is not equivalent to the original program P_1, but in turn, P_2 produces a program P_1, than we have a a hypercycle. Depending on the complexity of each of the individual programs, and their ontogeny, it may exhibit interesting autopoietic dependencies.

Data flow cycles. Each program within a multi-EVM environment fulfills its function in a narrow spectrum of data inputs, and produces its outputs again, in relatively narrow spectrum out of all possible outputs. For example a solver for “n-addition problem” cannot be given different input than it expects, otherwise it will not work as an “n-addition” problem solver. However, if the output of P_1 is connected to the input of P_2, and the output of P_2 to the input of P_1, then we have a cycle. If the cycle keeps the data flow within expected and desired ranges of values, we have an autocatalytic hypercycle. Together with the actual programs they represent an autopoietic system.

7 Self-organisation by Means of Evolutionary Computation

7.1 Requirements

There are some inherit properties that the self-adaptive and self-organising software system may exhibit. These properties facilitate effective processes to help and guide evolutionary mechanisms. Our current EVM implementation facilitates some of these properties.

Split and splice. It is desirable that different individual functional units are freely manipulated. It means that one can put different components together, and then split them apart, always producing valid functional units within the system. This is supported by the EVM. Each program can be cut in the middle, and the parts will always form valid programs. Programs can be joined together, always producing valid programs. Actually, any sequence of integers is a valid program in a EVM.

Cyclic behaviour. All individual components of the software system must carry out their activities in a cyclic manner. That means that the functionality is organised in such a way that tasks are repeated over and over again so that tuning, self-organisation, and adaptability can take place. If a given task were to be designed to be performed only once, there would be no room for improvement, since the given component would only have a single opportunity to perform.

Many agents on many levels. There are benefits from having many independent interactive components acting on many different levels. Reflection and recursion among components can facilitate shorter and more robust solutions to given tasks performed by components of the system. Some components will just perform tasks, some will monitor others performing tasks and provide necessary feedback for improvement, and others will improve the "improvers", etc.

The system must be open to external signals. This simply means that the system has to interact with the "outside". A software system which does not exchange any information with the environment "outside" the system itself cannot evolve into a more complex system than the original one. Without being exposed to new information the system can only refine itself, and is unable to acquire new capabilities. Such information must be provided from the outside environment.

7.2 Evolving Recursive Virtual Machines

The field of evolutionary computation is mainly based on experimentation, and so far it is primarily a trial and error approach. In light of all the advances in theoretical computer science and given the conceptual framework of recursive virtual machines, it is now possible to introduce a more systematic approach. Within EVM, each different evolutionary system is an example of a virtual machine, each language is an example of a different search space, and each system is an example of the interplay between different aspects of the hierarchical organisation.

Probably one of the closest existing systems using the concept of a virtual machine in the form of a hierarchy is the grammatical evolution system [28]. In this system, a top-level search is performed on strings of integers. A string containing integers is fed into a particular machine to produce a computer program coded in a particular language as output. This code is then fed as input to yet another machine, which in turn returns a final result. Each of the levels is relative to the level below it; this relativity means that the same top-level string of integers will produce a completely different result when used in combination with another machine. The top-level machine accepting the strings of integers is designed in such a way that it can "plug-in" to any possible second-level machine, and the model will still work. This is a human designed feature, but it is inspired by many naturally occurring phenomena. The multiple levels of indirect influences seems to be the most powerful mechanism at work here.

Instead of designing such machines, and all the indirection levels, by hand, we believe that with our approach this process can be automated, and the virtual machine suitable for a particular class of problems can be discovered automatically.

7.3 Seeds and Solution Growing

Let us take a grammatical evolution system [28] as an example of the solution growing concept. The solution for a problem at hand is effectively a proper hierarchy of machines (in this case a BNF-encoded language grammar) and a string of integers as a symbolically encoded solution, which we refer to as a *seed*. In the case of a grammatical evolution system, the hierarchy of machines is designed by a human programmer before the search for the proper seed is started. However, the hierarchy of machines needs to be discovered as well. sought-after solution itself.

In general, the solution to the problem (finding a computer program) will be a hierarchy of machines together with the seed. The actual computer program is then generated by feeding the seed through the system. In the case of grammatical evolution, speaking informally, the generation process is (in order): feeding the string of integers, generating the program listing, running the program for the given input, and then obtaining the final solution. The given input in this case depends on the "outer-level" virtual machine.

It is, however, possible to change or modify the machine hierarchy just before generating the computer program. If the hierarchy of machines, their connections and the initial states are subject to change, we refer to the process of generating a final solution as *solution growing*. In the case of searching for code, one can use the term *code growing* instead. It is possible, by varying the hierarchy of machines, to grow a valid solution from the same seed for a certain variation of the original problem. By simple re-mapping, one can achieve exactly the same result by varying the structure of the seed itself. This opens a new window of opportunities not yet used by the automatic code generation techniques. Again, it is a very commonly occurring phenomenon in nature.

Formally the idea of *code growing* is based on the notions of *bootstrapping* and *self-application*. This is analogous to more traditional compiler/interpreter bootstrapping and self-application [13].

8 Summary

An architecture of dynamic hierarchically organised virtual machines as a self-organising computing model has been presented. It builds on Turing-machine-based traditional models of computation. The model provides some of the necessary facilities for open-ended evolutionary processes in self-organising software systems. It allows stacking machines (vertical decomposition) in addition to more traditional functional hierarchical decomposition models. It can be used as a more systematic approach to different code generation techniques and self-adaptable software. Unlike existing models, the emerging levels of organisation can be either modelled directly as individual machines or can be indirectly captured for a formal analysis as a state of an individual machine.

Applications using the proposed architecture are possible and are planned as future work. Also, the formal model presented here allows for the preparation of an operational definition of a living system. However, further formalization of the framework is necessary, and is currently under investigation by the authors.

References

[1] Leon Brillouin. *Science and information theory*. Academic Press Inc., 1956.

[2] Gregory J. Chaitin. To a mathematical definition of 'life'. *ACM SICACT News 4*, pages 12–18, January 1970.

[3] Gregory J. Chaitin. Toward a mathematical definition of "life". In R. D. Levine and M. Tribus, editors, *The Maximum Entropy Formalism*, pages 477–498. MIT Press, 1979.

[4] Charles Darwin. *On the Origin of Species by Means of Natural Selection*. John Murray, 1859.

[5] Manfred Eigen and Peter Schuster. *The Hypercycle: A Principle of Natural Self-Organization*. Springer-Verlag, 1979.

[6] Gary William Flake. *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation*. MIT Press, January 31 2000.

[7] David B. Fogel, editor. *Evolutionary Computation – The Fossil Record*. IEEE Press, New York, USA, 1998.

[8] Jeff Fox. Multiple Stack Zero Operand Computers Today. *Free Software Magazine*, (05), 2002.

[9] Edward Fredkin. A new cosmogony: On the origin of the universe. In *PhysComp'92: Proceedings of the Workshop on Physics and Computation*. IEEE Press, 1992.

[10] Andrzej Gecow and Antoni Hoffman. Self-improvement in a complex cybernetic system and its implication for biology. *Acta Biotheoretica*, 32(1):61–71, 1983.

[11] John H. Holland. *Adaptation in Natural and Artificial Systems : An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, reprint edition edition, April 29 1992.

[12] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley Publishing Company, USA, 1979.

[13] Neil D. Jones. *Computability and Complexity: From a Programming Perspective*. MIT Press, 1997.

[14] Stuart A. Kauffman. *The origins of order: self-organization and selection in evolution*. Oxford Press, New York, USA, 1993.

[15] M. Kokar, K. Baclawski, and A. Eracar. Control theory-based foundations of self-controlling software. *IEEE Intelligent Systems*, pages 37–45, May-June 1999.

[16] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[17] Leonid A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

[18] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, New York, second edition, 1997.

[19] Lynn Margulis and Dorion Sagan. *What is life?* New York, Simon and Schuster, 1995.

[20] Humberto R. Maturana and Francisco J. Varela. Autopoiesis: The organization of the living. In Robert S. Cohen and Marx W. Wartofsky, editors, *Autopoiesis and Cognition: The Realization of the Living*, volume 42 of *Boston Studies in the Philosophy of Science*. D. Reidel Publishing Company, Dordrech, Holland, 1980. With a preface to 'Autopoiesis' by Sir Stafford Beer. Originally published in Chile in 1972 under the title De maquinas y Seres Vivos, by Editorial Univesitaria S.A.

[21] Alex C. Meng. On evaluating self-adaptive software. In Paul Robertson, Howie Shrobe, and Robert Laddaga, editors, *Self-Adaptive Software*, number 1936 in LNCS, pages 65–74. Springer-Verlag, Oxford, UK, April 17–19 2000. IWSAS 2000, Revised Papers.

[22] Charles H. Moore and Leach Goeffrey C. FORTH – A language for interactive computing. Technical report, Amsterdam NY: Mohasco Industries, Inc., 1970.

[23] L. E. Orgel. *The Origins of Life: Molecules and Natural Selection*. Wiley, New York, 1973.

[24] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, Inc., 1994.

[25] Ilya Prigogine and Isabelle Stengers. *From being to becoming*. San Francisco, W. H. Freeman., 1980.

[26] Martin Purvis, Stephen Cranefield, Geoff Bush, Daniel Carter, Bryce McKinlay, Mariusz Nowostawski, and Roy Ward. The NZDIS Project: an Agent-based Distributed Information Systems Architecture. In Jr. R.H. Sprague, editor, *In CDROM Proceedings of the Hawaii International Conference on System Sciences (HICSS-33)*. IEEE Computer Society Press, 2000.

[27] Justinian P. Rosca. *Hierarchical learning with procedural abstraction mechanisms*. PhD thesis, University of Rochester, Rochester, NY 14627, USA, 1997.

[28] Conor Ryan, J. J. Collins, and Michael O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *Proceedings of EuroGP 1998*, pages 83–96. Springer Verlag, 1998.

[29] J. Schmidhuber. The speed prior: a new simplicity measure yielding near-optimal computable predictions, 2002.

[30] Juergen Schmidhuber. A general method for incremental self-improvement and multiagent learning. In X. Yao, editor, *Evolutionary Computation: Theory and Applications*, chapter 3, pages 81–123. Scientific Publishers Co., Singapore, 1999.

[31] Juergen Schmidhuber. Optimal ordered problem solver. *Machine Learning*, 54:211–254, 2004.

[32] Erwin Schrödinger. *What is life? : the physical aspect of the living cell*. Cambridge, University Press, 1945.

[33] Claude E. Shannon and Warren Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, 1949.

[34] H. A. Simon. *The sciences of the artificial*. MIT Press, 1968.

[35] L. Spector and A. Robinson. Genetic programming and autoconstructive evolution with the Push programming language. *Genetic Programming and Evolvable Machines*, 3(1):7–40, 2002.

[36] Lee Spector. Hierarchy helps it work that way. *Philosophical Psychology*, 15(2):109–117, June 2002.

[37] Bill Venners. *Inside the Java Virtual Machine*. Mc-Graw Hill, second edition edition, 1999.

[38] John Luis von Neumann. The general and logical theory of automata. In A. H. Taub, editor, *John von Neumann – Collected Works*, volume V, pages 288–328. Macmillan, New York, 1963.

[39] John Luis von Neumann and Arthur W. Burks. Theory of self-reproducing automata, 1966.

[40] Michael D. Vose. *The Simple Genetic Algorithm: Foundations and Theory*. A Bradford Book, MIT Press, Cambridge, Massachusetts/London, England, 1999.

[41] Sewall Wright. Evolution in mendelian populations. *Genetics*, 16(3):97–159, March 1931.

Self-organising, Open and Cooperative P2P Societies – From Tags to Networks*

David Hales

Department of Computer Science, University of Bologna, Italy
dave@davidhales.com
www.davidhales.com

Abstract. For Peer-2-Peer (P2P) networks to realize their full potential the nodes they are composed of need to coordinate and cooperate, to improve the performance of the network as a whole. This requires the suppression of selfish behavior (free-riding). Existing P2P systems often assume that nodes will behave altruistically, but this has been shown to be far from the case. We outline encouraging *initial* results from a P2P simulation that translates and applies the properties of "tag" models (initially developed within social simulations) [8, 9] to tackle these issues. We find that a simple node rewiring policy, based on the tag dynamics, quickly eliminates free-riding without centralized control. The process appears highly scalable and robust.

1 Introduction

Open Peer-to-Peer networks (in the form of applications on-top of the internet) have become very popular for file sharing applications (e.g. Kazaa[1], Gnutella[2] etc). However, as has been shown [1] in such file sharing scenarios we find that a majority of users do not actually share their own files (they act selfishly). However, these networks are still popular because it only requires a minority to share high quality files for all to benefit - a small amount of altruism appears to be enough to support file-sharing applications. But what about P2P applications where high levels of altruism and cooperation *are* required (e.g. load balancing or cooperative routing)? How can selfish nodes to be discouraged? One solution is to have a closed system in which we can ensure that each node runs a particular peer application that is hard-coded to be cooperative. But this option precludes the benefits of open systems. In open systems the protocols are open so any node that understands the protocol can participate. This allows for truly decentralized control and freedom for innovation (new nodes with new kinds of behavior may enter the network). A desirable goal would be to have a network that could self-organize and adapt to a variety of tasks such that each node would benefit from the shared resources (bandwidth, processing, storage etc) that other nodes could offer.

---

* This work partially supported by the EU within the 6th Framework Programme under contract 001907 (DELIS).

[1] The Gnutella home page: http://www.gnutella.com/

[2] The Kazaa home page: http://www.kazaa.com

In order to archive such a desirable goal, don't we just need some very cleaver nodes? Or to put it another way, can we archive this goal "simply" by programming the peer nodes appropriately. Unfortunately there are some fundamental contradictions that need to be confronted when attempting to formulate how desirable collective action can be produced in open systems. We have to deal with the fact that each peer cannot make arbitrary *a priori* assumptions about the behavior of other peers. The assumptions (we can't avoid all assumptions) need to be as general as possible without being useless. Historically these issues have been studied and theoretically formulated within the social sciences (particularly Sociology, Political Science and Economics) with application to human social systems. Of course it would be foolish to believe that a set of generally agreed assumptions (concerning human social behavior) that a majority of social scientists would subscribe to could *ever* exist. There are several reasons for this including the complexity of human systems, the changing nature of human social organizations and behavior, the essentially political status that ideas applied to human society tend to acquire (particularly when those ideas are used to justify social action) and the fragmented nature of social science methodology. However, in the context of a kind of "worst case" set of assumptions, for engineering nodes in a P2P, we argue for that nodes should be seen as:

- In the network for what they can get out of it – selfish not altruistic
- Modify their behaviors to maximize individual benefit
- Have no (or limited) knowledge about other peers and the network in general

These assumptions imply a further one. That peers have some mechanism of determining how much they are benefiting from the system. This obviously would depend on the task domain e.g. for file sharing it would be some measure of how quickly requested files were found and downloaded or for group computing it might be a measure of processing resource donated by others[3].

Given these assumptions one fundamental problem is how to ensure that common resources (the commons) are utilized unselfishly for the benefit of all [15]. In the context of a P2P network we can view each peer as offering a set of commons resources. That is, peers through their actions, may offer resources to others or may not. Conversely peers make use of the resources offered by others. The fundamental problem is that given peers with the above assumptions under what conditions would peers converge towards sharing (benefiting all) as opposed to selfishly taking resources but offering none.

There are many possible ways to deal with these problems including the utilization of trusted 3rd parties, the generation and sharing of reputation information and behavioural strategies based on sanctions in future interactions [2]. However, in general such mechanisms demand high overheads in form of storage, processing and communication of information concerning on-going interactions and / or do not work in highly dynamic contexts where interactions will be predominantly with strangers. In its most condensed and abstracted form these kinds of scenario can be captured in the two player, single round Prisoners' Dilemma game where players represent peers (see below).

[3] It should be noted that in many real word task domains it is by no means clear what these measures might be. Certainly one can imagine situations in which no such simple measures could be determined. This would be particularly difficult for very delayed rewards.

Tags (see below) have recently been applied in these latter kinds of scenarios in the form of social simulations (with associated sociological interpretations - see [25]). Firstly, we will review relevant findings from the previous Tag simulations. Then we describe a simple simulation model of a P2P network and give some encouraging initial results. Finally we discuss the limitations of the model and the future direction we might take in order to address those limitations.

Along the way we attempt to present the method by which techniques have been imported from one kind of simulation scenario to another. The focus of the previous models was not on solving engineering problems; neither did those models deal with networks so the translation process was not straightforward.

2 What Are Tags?

Tags are markings or social cues that are attached to individuals (agents) and are observable by others [17]. They evolve like any other trait in a given evolutionary model. The key point is that the tags have no direct behavioral implication for the agents that carry them. Through indirect effects (such as biasing of interaction), however, they can evolve from initially random values into complex ever changing patterns that serve to structure interactions between individuals.

In the computational models presented here tags are modeled using some number (either a binary bit string, a real number or an integer). When agents interact they preferentially interact with agents possessing the same (or similar) tag value. One way to visualize this is to consider a population of agents partitioned between different colors. Each agent carries a single color. In a system with only three different possible tag values, we could think of this as each agent carrying a flag of red, green or blue. Agents then preferentially interact with agents carrying the same color (forming “interaction groups”). When agents evolve (using some form of evolutionary algorithm) they may mutate their tag (color). This equates to moving between interaction groups.

In the models presented here, tags take on many possible unique values (by say using a real number, there are many possible unique tags rather than just 3 colors) however, the basic process is the same – agents with the same tags preferentially interact and tags evolve like any other genotypic trait.

Another way to think of tags is that some portion of the genotype of an agent is visible directly in the phenotype but the other agents. In section 5 we give an outline algorithm of how tags are applied in a simple evolutionary system, firstly however, we introduce the Prisoners Dilemma game and some previous tag work.

3 The Prisoner’s Dilemma

The Prisoner's Dilemma (PD) game captures a scenario in which there is a contradiction between collective and self-interest. Two players interact by selecting one of two choices: Either to "cooperate" (C) or "defect" (D). For the four possible outcomes of the game players receive specified payoffs. Both players receive a reward payoff (R) and a punishment payoff (P) for mutual cooperation and mutual defection respectively. However, when individuals select different moves, differential payoffs of temptation (T) and sucker (S) are awarded to the defector and the cooperator respectively. Assuming that neither player can know in advance which move the other will make and wishes the maximize her own payoff, the dilemma is evident in the ranking of payoffs: $T > R > P > S$ and the constraint that $2R > T + S$. Although both players would prefer T, only one can attain it. No player wants S. No matter what the other player does, by selecting a D move a player ensures she gets either a better or equal payoff to her partner. In this sense a D move can't be bettered since playing D ensures that the defector cannot be suckered. This is the so-called "Nash" equilibrium for the single round game. It is also an evolutionary stable strategy for a population of randomly paired individuals playing the game where reproduction fitness is based on payoff. So the dilemma is that if both individuals selected a cooperative move they would both be better off but both evolutionary pressure and game theoretical "rationality" selected defection.

For a detailed treatment of the PD, its relationship to social and evolutionary science and a serious, original and thought provoking analysis of the evolution of non-suboptimal behavior from selfish interactions see [16][4].

4 Previous Tag Models

There have been a number of tag models implemented previously. All demonstrate higher-than-expected levels of cooperation and altruism from seeming selfish individuals. All implement evolutionary systems with assumptions along the lines of the replicator dynamics (i.e. reproduction into the next generation proportional to utility in the current generation, no "genetic-style" cross-over operations but low probability mutations on tags and strategies during reproduction).

Riolo [22] gave results of expansive and detailed studies applying tags in a scenario where agents played dyadic (pair wise) Iterated Prisoner's Dilemma games (IPD). Tags (represented as a single real number) allowed agents to bias their partner selection to those with similar tags (probabilistically). He found that even small biases stimulated high levels of cooperation when there were enough iterations of the game with each pairing.

In Riolo et al [23] a tag model was applied to a resource-sharing scenario in which altruistic giving was shown to emerge. Agents were randomly paired (some number of times) and decided if to give resources or not. The decision to give was based on tag similarity mediated by a "tolerance gene" as well as the "tag gene" (both represented as real numbers). The utility to the receiving agent of any given resource was greater than to that of the giving agent. It was shown that if each agent was paired enough times in each generation and the cost / benefit ratio was low enough then high levels of cooperation were found.

In Hales and Edmonds [12] tags were applied to a simulated robot coordination scenario producing high levels of cooperative help giving.

In Hales [8] a tag model was applied to a single round PD. Again interaction was dyadic. Tags were represented as binary strings. Pairing was strongly biased by tag

[4] Also see information about he PD online at the wonderful "Principia Cybernetica Project website: http://pespmc1.vub.ac.be/PRISDIL.html

identity (rather than probabilistic similarity). In this model very high levels of cooperation were produced between strangers in the one shot game. A refinement of this model in [11] showed how the same result could be produce with tags represented as real numbers so long as the probability of mutation being applied to the tag is higher than that applied to the strategy (by about one order of magnitude).

For the purposes of this paper we will now outline in a little more detail these latter models applied to the PD.

5 Tags and the PD

In [8] a model is presented of agents playing the PD in pairs in a population with no topological structure (other than tag based biasing of interaction). The mode is composed of very simple agents. Each agent is represented by a small string of bits. On-going interaction involves pairs of randomly selected agents playing a single round of PD. Agent bits are initialized at random. One bit is designated as the PD strategy bit: agents possessing a "1" bit play C but those possessing a "0" bit play D. The other bits represent the agents tag. These bits that have no direct effect on the PD strategy selected by the agent but they are observable by all other agents. Below is an outline of the simulation algorithm used:

```
LOOP some number of generations
    LOOP for each agent (a) in the population
        Select a game partner agent (b) with the same tag (if possible)
        Agent (a) and (b) invoke their strategies and get appropriate payoff
    END LOOP
    Reproduce agents in proportion to their average payoff
    With low probability, mutate the tag and strategy of each reproduced agent
END LOOP
```

Agents are selected to play a single-round of PD not randomly but based on having the same tag string. If an agent can find an individual with the same tag string as its own in the system it will play PD against that agent. If it cannot then it plays against some randomly chosen partner. Agents are reproduced probabilistically in proportion to average payoff they received (using roulette wheel selection).

Extensive experimentation varying a number of parameters showed that if the number of tag bits is high enough[5] (in this case we found 32 tag bits for a population of 100 agents to be sufficient with a mutation rate of 0.001 and PD payoffs of T=1.9,

[5] In a more recent model Hales (forthcoming) we demonstrate that the requirement for many tag bits was because this effectively increased the mutation rate applied to the tag as a whole (since mutation was applied to each bit with the same probability as mutation was applied to the single strategy bit).

R=1, P=S=0.0001[6]) then high levels of cooperation quickly predominated in the population[7].

More interesting still, if all the agents are initially set to select action D (as opposed to randomly set) then the time required to achieve a system where C actions predominate is found to monotonically decrease as population size increases. This is an inverse scaling phenomena: the more agents, the better. Additionally the fact that the system can recover from a state of total D actions to almost total C actions (under conditions of constant mutation) demonstrates high robustness. The tag-based model produces an efficient, scalable and robust solution – based on very simple individual learning methods (modeled as reproduction and mutation).

5.1 How Tags Work

We have described this model because it seems to offer up a method for achieving three important properties in a simple (PD) task domain: efficiency, scalability and robustness. But how do tags produce this seemingly magical result? The key to understanding the tag process is to realize that agents with the same tag strings can be seen as forming a sort of "interaction group". This means that the population can be considered as a collection of groups. If a group happens to be entirely composed of agents selecting action C (a cooperative group) then the agents within the group will outperform agents in a group composed entirely of agents selecting action D (a selfish group). This means that more agents will copy the behavior of cooperative groups than selfish groups. By copying the behavior and the tags of those who perform well, agents are essentially joining groups that are cooperative. However, if an agent happens to select action D within a cooperative group then it will individually outperform any C acting agent in that group and, initially at least, any other C acting agent in the population – here the T payoff is 1.9 where as the best a C acting agent can do is R = 1.

However, by others copying such an agent (i.e. the agent reproducing copies of itself) the group becomes very quickly dominated by D acting agents and therefore the relative advantage of the lone D acting agent is lost – the group snuffs itself out due to the interaction being kept within the group. So by selecting the D action an agent destroys its group very quickly (remember groups are agents all sharing the same tag string). Figure 1 visualizes this group process in a typical single run. Each line on the vertical axis represents a unique tag string. Groups composed of all C action agents are shown in light gray (Coop), mixed groups of C and D agents are dark gray and groups composed of all D are black.

The tag mechanism, then, precipitates a kind of "group selection" process in which those groups which are more cooperative tend to predominate but still die out as they are invaded by mutant D acting agents. In a real sense the groups compete for resources despite the fact that evolution only occurs at the individual level and the agents don't even know they are in such a group. In this system, the agents don't die, just the particular groupings (based on sharing the same tag string) change. By

[6] P and S were set to the same small value for simplicity. If a small value is added to P (enforcing T > R > P > S) results are not significantly changed.

[7] If tags are removed from the model and pairing for game playing is completely random then the population quickly goes to complete defection (the Nash equilibrium for the single-round PD).

constantly changing tag strings (by reproduction of those with higher utility) the agents produce a dynamic process that leads to high levels of C actions. In other words, the population as a whole contains a lot of cooperation occurring within a constantly changing system of groups, even though each agent is acting without any knowledge of the group structure and there is no central coordination of the groups. Typically cooperative interactions in the model reach over 90% of all interactions (over 100,000 cycles).

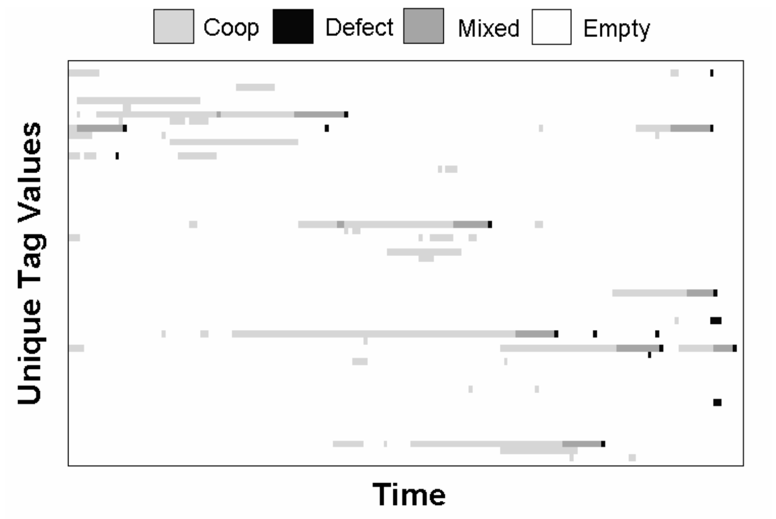


Fig. 1. Visualization of 200 cycles (generations) from a single simulation run showing cooperative groups coming into and going out of existence. See the text for a full explanation

6 From Tags to Networks

The underlying mechanism driving cooperation within the tag simulation is the formation and dissolution of sharply delineated groups of agents (identified by sharing the same tag). Each agent could locate group members from the entire population. Each member of the group had an equiprobable chance of interacting with any agent in the population sharing the same tag. In this sense each agent could determine which agents were in their group.

If we assume a sparse P2P network in which each node (peer) knows of some small number of other nodes (neighbors) and those neighborhoods are highly interconnected (clustered) such that most neighbors share a large proportion of other neighbors then we have something similar to our tag-like groupings. Instead of a tag (a marker) we have an explicit list of neighbors. In a highly clustered network the same list will be shared by most of the neighborhood. In this sense you can visualize the table of known peers stored in each agent (its neighbors) as the tag. It is shared by the group and is the key by which the group can directly interact with each other. To this extent it defines a group boundary. A nice feature of this also is that it is a kind of watertight method of isolating nodes into neighborhoods (for direct interaction) since a node cannot go directly interact with another node that it does not know of.

In our initial model we did not restrict the size of the neighborhood (i.e. networks could be non-sparse) but we wired the initial network topology as "small world" (i.e. highly clustered regular lattice but with random rewiring with low probability – see [27]). Also we set the tag mutation probability (changing the neighborhood to a single randomly chosen neighbor) to an order of magnitude higher than the strategy mutation probability (flipping the strategy bit). We found later that we did not need to wire a small world and that any initial wiring self-organized to high clustering over time. Sparse random wiring was finally chosen for simplicity.

We investigate only direct interactions between neighbors in this model. In a sense this is all that can ever happen in P2P systems. Indirect interactions between nodes that do not share neighborhoods have to mediate by direct interactions between intermediate nodes. Essentially, one can view all interaction as with neighbors (even if those neighbors are actually proxies for other more remote nodes). If cooperation can be established between the majority of neighborhoods in a network then it follows that any pair of nodes in the network that are connected will have a good chance of being able to find a *path of cooperation* through the network.

In order to capture this kind of neighborhood interaction in the simplest possible way we have each node in the network play a single round of PD (see above) with a randomly chosen neighbor. No information is stored or communicated about past interactions and the topology is not fixed (see below).

6.1 Neighbor Lists as Tags, Mutation as Movement

In the tag model change was produced over time by mutation and differential reproduction based on average payoff. How can these be translated into the network?

In our network model we do not view nodes as "reproducing" in a biological sense or cultural sense. However, it is consistent with or initial assumptions (see above) that nodes may relocate to a new neighborhood in which a node is performing better than itself. That is, we assume that periodically nodes make a comparison of their performance against another node randomly chosen from the network[8]. Suppose node (i) compares itself to (j). If (j) has a higher average payoff than (i) then (i) disregards its neighbor list and copies the strategy and neighbor list of (j) also adding (j) into the list. This process of copying can be visualized as movement of the node into the new neighborhood that appears more desirable.

Mutation in the tag model was applied after reproduction. Each bit of the tag and the strategy was mutated (flipped) with low probability. Since we are using the same one bit strategy we can apply mutation to the strategy in the same way. We therefore flip the strategy bit of a node with low probability immediately after reproduction (the movement to a new neighborhood as described above). Since we treat the list of neighbors in each node as the tag a mutation operation implies changing the list is some way. But we can't simply randomly change the list; we need to change the list in such a way as to produce an effect with closely follows what happens when mutation is applied in the tag model. In that model, tag mutation tended to give agents unique tags – i.e. tags

[8] Currently we do not model the process of finding this "out-group" node. We assume that the network could provide the service – but this might be a problem (see conclusion).

not shared by other agents at that time. However, in the model agents could interact with a randomly chosen agent with non-matching tags if none existed with identical tags. In this way tag mutation lead to the founding of new tag groups.

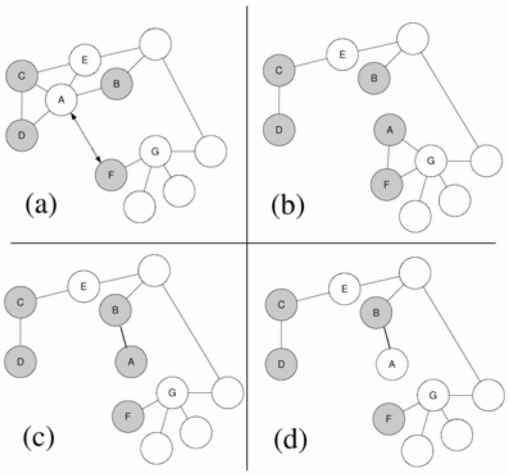


Fig. 2. An illustration of "replication" and "mutation" as applied in NetWorld. Shading of nodes represents strategy. In (a) the arrowed link represents a comparison of utility between A and F. Assuming F has higher utility then (b) shows the state of the network after A copies F's links and strategy and links to F. A possible result of applying mutation to A's links is shown in (c) and the strategy is mutated in (d)

In the network model we don't want to isolate the node completely from the network otherwise it will not be able to interact at all. However, we don't want to move into an existing neighborhood (as with reproduction) but rather to do something that may initiate the founding of a new neighborhood. So we pragmatically express tag mutation as the replacement of the existing neighbor list with a single neighbor drawn at random from the network.

We now have our analogues of reproduction and mutation for the network model. Reproduction involves the nodes copying the neighbor lists and strategies of others obtaining higher average scores. Mutation involves flipping the strategy with low probability and replacing the neighbor list with a single randomly chosen node with a low probability (see figure 2). In the next section we outline out new network model – NetWorld[9].

7 The NetWord Model

The NetWorld model is composed of a set N of nodes (or peers). Each node stores a list of other nodes it knows about (we term this the neighbor list). In addition to the

[9] There are many other ways tags could be translated into networks. For example, agents could move around the network between nodes carrying tags or agents sharing a node could be seen as sharing a tag. We hope to explore some of these variations in the future.

neighbor list each node stores a single strategy bit indicating if it is to cooperate or defect in a single round game of the PD. Neither the strategy bit nor the list is normally visible to other nodes. Initially nodes are allocated a small number of neighbors randomly from the population. Periodically each node selects a neighbor at random from its list and plays a game of PD with it. Each node plays the strategy indicated by its strategy bit. After a game the relevant payoffs are distributed to each agent. Periodically pairs of randomly chosen nodes (i, j) compare average payoffs. If one node has a lower payoff then the strategy and neighbor list from the other node is copied (effectively moving the lower scoring node to a new neighborhood). Mutation is applied to both the strategy and the neighbor table with probability m. Mutation of the strategy involves flipping the bit. Mutation of the neighbor list involves clearing the list and replacing it with a single randomly chosen node from the population. Below is an outline of the NetWorld simulation algorithm:

```
LOOP some number of generations
    LOOP for each node (i) in the population
        Select a game partner node (j) randomly from neighbor list
        Agent (i) and (j) invoke their strategies and get appropriate payoff
    END LOOP
    Select N/2 random pairs of nodes (i, j)
    Copy higher scoring node into lower scoring node
    Apply mutation to tag and strategy of each reproduced node with probability
    m
END LOOP
```

The neighbor lists are limited in size to a small number of entries. The entries are symmetrical between neighbors (i.e. if node (i) has an entry for node (j) in its list then node (j) will have node (i) in its list). If a link is made to a node that has a full neighbor list then it discards a randomly chosen neighbor link in order to make space for the new link. Also if a node is found to have no neighbors when attempting to play a game of PD (this can happen if neighbors have moved away) then a randomly chosen node is made a neighbor.

7.1 Initial Results

Figure 3 gives some initial results. In these experiments the mutation rate m = 0.001 and the PD payoffs were as per the previously described tag model (see above). In all the results given here we start he population from complete defection and wired the initial network topology by given each node a fixed small number of links to randomly chosen nodes .

We tried increasing the mutation rate applied to the tags (i.e. the neighbor list) by an order of magnitude and this reduced the time to cooperation an increased the scalability. Over all sensible parameter values so far tried we have found extremely encouraging results. Of particular surprise was the speed of convergence to high cooperation. Even when $N=10^5$ and all nodes were initially started with D (defect) strategies it took only approximately 140 cycles on average to achieve 99% of nodes utilizing the C (cooperate) strategy.

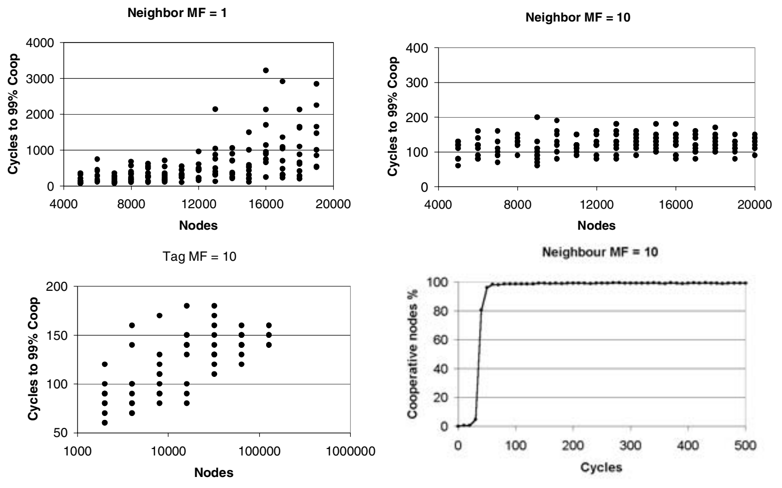


Fig. 3. Charts show initial results from the model. The top charts show the time taken in cycles for the network to reach a state where 99% of nodes are cooperators for different sizes of network. Each dot is an individual run. In the right chart mutation on the neighborhood (tag) is 10 times that on the strategy. The bottom left chart shows extended results on a log scale. The bottom right chart shows a typical single run time series (with a 10,000 node network)

We hoped to find the reverse scaling cost and differential mutation characteristics identified in previous non-network based tag models [8, 11]. The reverse scaling does not appear to be present. The application of differential (higher) mutation on tags (neighbor tables), appears to bring the upper-bound (of time to cooperation) down to log(nodes) improving scalability. As stated previously, we did not find that the initial form of the network made much difference to the results. Even starting the network fully connected or completely unconnected (all with no links) did not change our results significantly. This seems to suggest a high level of robustness – something we are interesting in achieving. Again much more experimentation and analysis is needed.

8 Discussion and Related Work

Only after our network translation of tags as the dynamical rewiring of the network (as nodes seek to improve their neighborhoods) did we realize the wealth of material that becomes relevant. Specifically our model now bears a very close comparison with that given by Zimmermann et al [28]. Zimmermann et al start with a network that is interpreted as representing a social network (it's a social simulation). Agents play PD with all their neighbors at each time step. Defecting agents then selectively replace existing links to other defectors with randomly chosen nodes (this is done probabilistically). They find steady states (of high cooperation) in which long chains of cooperators are formed in which founder "leader" nodes are highly important to stability. Prior to steady states there are oscillations in the levels of cooperation. They use synchronous updating throughout and do not include noise (in the form of spontaneous change of strategy) in most of their analysis. However they do study the effect of a single noise event on significant nodes (they call leaders) and show that in their model even a single mutation event (changing the strategy of a single node) can completely wipe out cooperation in the entire network for many cycles. This is one of the major findings of the model, that the steady states become highly sensitive to changes in single key nodes. They also find that their network when in the steady state tends to be highly disconnected leading to a number of component sub-networks forming. In their model the network was populated with 60% cooperative agents. Since they have no spontaneous adaptation or noise on strategies their model would never escape from a global minima of all nodes defecting. So although the models are very similar theirs has several properties that we would want to avoid in our model[10] such as not being able to recover from total defection and the high sensitivity of the network to the behavior of a very small number of "leaders". Additionally in their model agents needed a little more local information (i.e. the strategies of the other agents so they could preferentially break links).

The fitness related preferential rewiring in our model obviously has linkages with the preferential linking ideas expressed by Barabási [3]. In subsequent experimentation with our model we hope to characterize the kinds of networks that are being formed over time. It will be of interest to compare these to the kinds of naturally occurring networks and rewiring methods that have been studied by Barabási. We hope that we may be able to utilise these theoretical and empirical contributions to increase understanding and efficient of our model with respect to tougher task domain.

It should be mentioned that Watts [27] looked at the results of playing the repeated PD game (rather than the single round game) on various fixed network topologies. Reproduction of strategies was within the local neighborhood. Various repeated strategies were tested. On the whole no the simulation results presented showed that it was difficult to get cooperation to dominate the network even with repeated strategies like tit-for-tat as popularized by Axelrod [2]. He found that some kinds of fixed small world networks could help sustain cooperation.

Interestingly Cohen et al [4] examined the results of extensive experimentation where *both* tags and networks (fixed random) were examined for their contribution to promoting cooperation in a PD scenario. Again only the *repeated* game was examined (not the single round game). However, they did not *combine* tags and networks but rather compared independent simulations.

A recent paper of Sun & Garcia-Molina [21] applies "incentives" within a simulated P2P file-sharing scenario in order to encourage selfish nodes to share resource. Their model relies on repeated interaction with nodes updating weights between on links to neighbors. Although they have not yet tested their system in a evolving environment, they don't require utility comparison between nodes since nodes simply update their weights based on service gained and then share out service supplied proportionate to weight (a kind of tit-for-tat [2]). This means that a selfish node quickly gets less and less service from it's neighbors. In future work we hope to apply the query scenario given by Sun & Garcia-Molina to our more dynamic scenario.

[10] However, we found it intriguing to consider if we would have reached a similar model to NetWorld if we had *started* with the Zimmermann et al model and attempted to make it more robust.

9 Conclusion

At this early stage our conclusion contains more questions than answers. However, the basic result of these initial experiments is that high-levels of cooperation can be produced and sustained in very large P2P by following this simple re-wiring and mutation scheme inspired by results from previous tag models. It appears we have been successful in importing the tag like dynamics into the network.

As stated previously these are preliminary results from a preliminary model and there are a number if outstanding issues before we can refine the model to incorporate more realistic P2P-like conditions. For example, we don't model the maintenance of up-to-date neighbour tables in the face of unstable links and nodes. Neither do we model the underlying process of finding random nodes in the network. This shortcut needs to be modeled using the P2P itself to supply new such nodes for the purposes of reproduction and mutation. What would be important here would be to find an efficient scalable way (probably therefore non-uniformly random) to supply nodes that allowed cooperation to form. We hope to test our results on a simulated version of something like NEWSCAST [18] - a highly robust and scalable P2P infrastructure.

We have yet to properly analyze the dynamics in the model. What kinds of networks topologies are being formed? We currently don't know how average path lengths, clustering and other topological features of the network evolve over time. It may even be the case the network regularly breaks into a number of disconnected components[11]. This would be serious problem if such breaks persist and are numerous since this would limit the possible size of the P2P network. All we currently know is that when cooperation is low the average degree of each node (size of the neighbor list) is near maximum but is lower when cooperation is high. This does not tell us too much.

The PD task domain although useful is a rather impoverished task domain. As an initial proof of concept it shows that at least some kinds of social dilemma can be solved. But the behaviors (PD strategies) and coordination required is trivial (although the dilemma itself is not trivial). We would therefore like to extend the simulation model to include more realistic kinds of task such as those requiring the coordination of a number of peers performing specialized functions.

A more important general issue raised by this kind of work[12] (in the context of applying models originating in the assumptions of evolutionary theory) is the assumption that all nodes behave as bounded optimizers. In our model we do not allow for nodes that simply "whitewash" (i.e. never adapt but just defect) or nodes that don't move, or worse nodes that move very fast but never adapt their strategy. This assumption does not hold in many situations and we need to explore alternative mechanisms to make model robust to these possibilities.

[11] Very recent work (since the first review of this paper) does indeed show that the network regularly breaks into disconnected components – which raises issues of if this mechanism would support long range routing tasks. However, the network is in constant flux (in a similar way to the groups in figure 1) with cliques forming and dissolving so this may be possible over some temporal window.

[12] And pointed out by a perceptive reviewer of the initial draft of this paper!

Acknowledgements

Thanks go to Mark Jelasity for pointing out some of the recent models that bear close comparison to this one. Also, along with Ozalp Babaoglu and Alberto Montresor, for writing clear and readable papers about P2P systems that have helped me in beginning this line of work. Thanks to Bruce Edmonds and Scott Moss at the Centre for Policy Modelling (CPM) in Manchester where much of the initial tag work was graciously supported and encouraged. Thanks also to the reviewers of the first draft of this paper. Their generous comments and encouragement were invaluable.

References

[1] Adar, E. and Huberman, B. A. (2000) Free Riding on Gnutella. First Monday Volume 5, No. 10, (http://www.firstmonday.dk/issues/issue5_10/adar/index.html).

[2] Axelrod, R. (1984) The Evolution of Cooperation, Basic Books, New York.

[3] Barabási, Albert-Lázló (2002) Linked: The New Science of Networks, Cambridge, MA: Perseus Publishing

[4] Cohen, M., Riolo, R. and Axelrod, R. (1999) The emergence of social organization in the prisoner's dilemma: how context-preservation and other factors promote cooperation. Santa Fe Institute Working Paper 99-01-002.

[5] Davis, L. (1991) Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York.

[6] Di Marzo Serugendo, G. "Engineering Emergent Behaviour: A Vision", Invited Talk. Multi-Agent-Based Simulation III. 4th International Workshop, MABS 2003 Melbourne, Australia, July 2003, D. Hales, B. Edmonds, E. Norling, J. Rouchier (Eds), LNAI 2927, Springer-Verlag, 2003.

[7] Edmonds, B. and Hales, D. (2003) Replication, Replication and Replication - Some Hard Lessons from Model Alignment. Journal of Artificial Societies and Social Simulation 6(4).

[8] Hales, D. (2000), Cooperation without Space or Memory: Tags, Groups and the Prisoner's Dilemma. In Moss, S., Davidsson, P. (Eds.) Multi-Agent-Based Simulation. Lecture Notes in Artificial Intelligence, 1979:157-166. Berlin: Springer-Verlag.

[9] Hales, D. (2001) Tag Based Cooperation in Artificial Societies. PhD Thesis (Dept. Of Computer Science, University of Essex, U.K. 2001).

[10] Hales, D. (2002) Evolving Specialisation, Altruism and Group-Level Optimisation Using Tags. In Sichman, J. S., Bousquet, F. Davidsson, P. (Eds.) Multi-Agent-Based Simulation II. Lecture Notes in Artificial Intelligence 2581:26-35 Springer Verlag. Berlin.

[11] Hales, D. (forthcoming), Change Your Tags Fast! - A necessary condition for cooperation? Submitted to the MAMABS workshop at AAMAS 2004.

[12] Hales, D. and Edmonds, B. (2003) Evolving Social Rationality for MAS using "Tags", In Rosenschein, J. S., et al. (eds.) Proceedings of the 2nd International Conference on Autonomous Agents and Multiagent Systems, Melbourne, July 2003 (AAMAS03), ACM Press, 497-503

[13] Hales, D. and Edmonds, B. (2004) Can Tags Build Working Systems? - From MABS to ESOA, In Di Marzo Serugendo, G.; Karageorgos, A.; Rana, O.F.; Zambonelli (eds.) Engineering Self-Organising Systems - Nature-Inspired Approaches to Software Engineering. Lecture Notes in Artificial Intelligence 2977, Springer, Berlin.

[14] Hamilton, W. D. (1964) The genetical evolution of social behaviours, I and II. J. Theor. Biol. 7, 1-52.

[15] Hardin, Garrett (1968) "The Tragedy of the Commons,", Science, 162:1243-1248.

[16] Heylighen F. (1992) : "Evolution, Selfishness and Cooperation", Journal of Ideas, Vol 2, # 4, pp 70-76.

[17] Holland, J. (1993) The Effect of Lables (Tags) on Social Interactions. Santa Fe Institute Working Paper 93-10-064. Santa Fe, NM.

[18] Jelasity, M., Montresor,A., and Babaoglu, O. (2004) A modular paradigm for building self-organizing peer-to-peer applications. , In Di Marzo Serugendo, G.; Karageorgos, A.; Rana, O.F.; Zambonelli (eds.) Engineering Self-Organising Systems - Nature-Inspired Approaches to Software Engineering. Lecture Notes in Artificial Intelligence 2977, Springer, Berlin.

[19] Nowak, M. & May, R. (1992) Evolutionary Games and Spatial Chaos. Nature, 359, 532-554.

[20] Nowak, M. & Sigmund, K. (1998) Evolution of indirect reciprocity by image scoring. Nature, 393, 573-557.

[21] Qixiang Sun & Garcia-Molina (2004) SLIC: A Selfish Link-based Incentive Mechanism for Unstructured Peer-to-Peer Networks. In Proceedings of the 24th IEEE international Conference on Distributed Systems (March 2004). IEEE computer Society.

[22] Riolo, R. (1997) The Effects of Tag-Mediated Selection of Partners in Evolving Populations Playing the Iterated Prisoner's Dilemma. Santa Fe Institute Working Paper 97-02-016. Santa Fe, NM.

[23] Riolo, R. L., Cohen, M. D. & Axelrod, R. (2001) Evolution of cooperation without reciprocity. Nature 414, 441-443

[24] Roberts, G. & Sherratt, T. N. (2002) Nature 418, 449-500

[25] Sigmund, K. and Nowak, A, M. (2001) Tides of Tolerance. Nature 414, 403-405.

[26] Trivers, R. (1971) The evolution of reciprocal altruism. Q. Rev. Biol. 46, 35-57.

[27] Watts, D. (1999) Small Worlds: The Dynamics of Networks between Order and Randomness. Princeton University Press. Princeton, New Jersey.

[28] Zimmermann, M. G., Victor M. Egufluz and Maxi San Miguel (2001) Cooperation, adaptation and the emergence of leadership in `Economics with Heterogeneous Interacting Agents', pp. 73-86, A. Kirman and J.B. Zimmermann (Eds.), Springer, Berlin.

Self-organizing Spatial Shapes in Mobile Particles: The TOTA Approach

Marco Mamei, Matteo Vasirani, and Franco Zambonelli

Dipartimento di Scienze e Metodi dell'Ingegneria,
Università di Modena e Reggio Emilia – Italy
{mamei.marco, vasirani.matteo, franco.zambonelli}@unimo.it

Abstract. We present a programming approach to let a multitude of simple mobile computational "particles" (i.e. sorts of tiny mobile robots) to self-organize their respective locations to assume a coherent global formation (i.e. shape). The problem has a variety of applications in mobile robotics, modular robots, sensor networks, and computational self-assembly. Here we show how the TOTA ("Tuples On The Air") middleware can be effectively exploited to enable self-organization of spatial shapes in mobile particles with minimal capabilities. The key idea in TOTA is to rely on spatially distributed tuples, spread across the network, to drive particles' movements and activities. Several experiments are reported showing the effectiveness of the approach.

1 Introduction

In the near future, micro-electro-mechanical systems will be embedded in the fabric of our everyday life. They will be able to interact with the physical world to provide an endless range of activities and services. In this perspective, we envision the possibility of exploiting these technologies to build sorts of multi-cellular computational organisms, made up of millions of interacting autonomous computational particles, capable of assembling and dynamically re-assembling themselves into a variety of complex shapes (as the T1000 robot in the Terminator 2 movie).

Although the hardware technology for these kinds of scenarios is rapidly maturing, software engineering practices have remained more or less the same since structured design and distributed programming methodologies were introduced: components are coupled at design time by fixed interaction patterns. Although simple, this approach turned out to be really brittle and fragile, not being able to cope with reconfiguration and faults. On the contrary, application components should be able to coordinate their activity patterns autonomously at run-time, despite, and possibly taking advantage, of environment dynamics and unexpected situations.

In general, the critical task is to identify appropriate (self)organization principles and programming methodologies for controlling the overall behavior of such complex systems. In particular, our goal is to study how and to which extent a group of simple mobile autonomous particles can be programmed to coordinate their respective movements and create variety of global shapes. Apart from futuristic nano-technology scenarios such as computational self-assembly [12] and the T-1000 vision, such a problem has more practical short-term applications, e.g. coordinate the movements of navigator-equipped cars or that of a PDA-equipped rescue team [10], enforcing self-deployment of sensor networks [4] and robots in a landscape [1, 5].

Biological organisms, achieving coherent, reliable and complex behavior from the local cooperation of large numbers of identically "programmed" cells, are of course the most natural source of inspiration for all these kinds of problems. In particular, the diffusion of chemicals among cells and the possibility for cells to be driven in their behavior by the locally sensed gradients of diffused proteins ("morphogen gradients") [3], due to its simplicity, seems suitable for applications to the problem of pattern formation in simple computational particles.

In our research, we developed a general purpose middleware called "Tuples On The Air" (TOTA). TOTA is an extremely lightweight middleware suitable for resource limited devices. TOTA provides abstractions and mechanism to support the creation of distributed overlay data structures spread across a mobile network. Such overlay data structures have the property of self-maintaining their intended distribution despite network dynamism (due to nodes connections, disconnections and movements).

TOTA tuples can easily resemble morphogen gradients in a network of mobile particles and thus the abstractions promoted by TOTA seem very suitable to manage the spatial self-organization in an ensemble of mobile particles.

A large number of papers deal with pattern formation in mobile robots [1, 5, 17, 19], and some exploit approaches somewhat similar to the one of morphogen gradients [12, 16, 18]. The key contribution here is to show how a variety of patterns (from regular to non-regular ones, also involving differentiation in particles) can be achieved even in the absence of those capabilities (e.g., global perception, distance and direction sensing) that are required by most other approaches.

This paper is organized as follows. Section 2 overviews the key characteristics of the TOTA middleware. Section 3 introduces the main steps required to let the robots self-organize their activities and arrange in specific global shapes. Section 4 presents several examples of global shapes we had been able to achieve. Section 5 discusses related works. Finally, Section 6 concludes and outlines future works.

2 The Tuples on the Air Approach

TOTA is composed by a dynamic ad-hoc wireless network of possibly mobile nodes, each running a local version of the TOTA middleware.

Upon the distributed space identified by the dynamic network of TOTA nodes, each peer is capable of locally storing tuples [6] and letting them diffuse through the network. Tuples are injected in the system from a particular node, and spread hop-by-hop accordingly to a specified propagation rule. Specifically, in TOTA, overlay data structures have been realized by means of distributed tuples ***T=(C,P)***, characterized by a content ***C*** and a propagation rule ***P***. The content ***C*** is an ordered set of typed fields representing the information carried on by the tuple. The propagation rule ***P*** determines how the tuple should be distributed and propagated in the network. This includes determining the "scope" of the tuple (i.e. the distance at which such tuple should be propagated and possibly the spatial direction of propagation) and how such propagation can be affected by the presence or the absence of other tuples in the system. In addition, the propagation rules can determine how tuple content should change while it is propagated to actually create distributed data structures.

The spatial structures induced by tuples propagation must be maintained coherent despite network dynamism. For instance, when new nodes get in touch with a network, TOTA automatically checks the propagation rules of the already stored tuples and eventually propagates the tuples to the new nodes. Similarly, when the topology changes due to nodes' movements, the distributed tuple structure changes to reflect the new topology (see figure 1).

Given these features, as it will be better described in the next sections, TOTA tuples are particularly suitable in implementing distributed data structures mimicking morphogen gradients.

From the application components' point of view, executing and interacting basically reduces to define and inject tuples in the network (*inject* method) and to read local (*read* method) and one-hop neighbor (*readOneHop* method) tuples via a pattern-matching mechanism. TOTA provides a compact API to perform these operations.

From an implementation point of view, we developed a first prototype of TOTA running on Linux IPAQs equipped with 802.11b WLAN and Java (J2ME, CDC, Personal profile). Moreover, we have implemented an emulator to analyze TOTA behavior in presence of hundreds of nodes.

In this paper we used the TOTA emulator to identify and test several algorithms to let a group of nodes, mimicking the mobile particles, to self-organize their spatial distribution. Further details on the TOTA middleware can be found in [11].

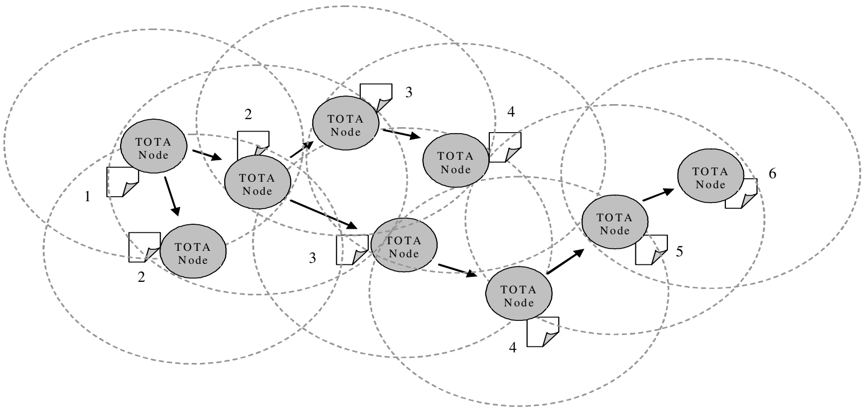


Fig. 1. The general scenario of TOTA: application components live in an environment in which they can inject tuples that autonomously propagate and sense tuples present in their local neighborhood. The environment is realized by means of a peer-to-peer network where tuples propagate by means of a multi-hop mechanism

3 Self-organize Spatial Shapes with TOTA

In this section, we first describe our model for particles and how they can exploit morphogen gradients towards pattern formation. Then we present related approaches and possible objections to our model.

3.1 Mobile Particles

To be "compliant" with foreseeable future micro and nano computer-based scenarios, we focus on particles with minimal capabilities. Specifically:

1. Particles are autonomous (i.e. have a separate thread of execution and control) and are equally programmed (i.e. they run the same code). Differentiation in their activities – if needed – must be established run-time on the basis of the data (e.g. morphogen gradients) they perceive.
2. In any case, each particle is provided with a random number generator enabling an additional simple form of symmetry breaking and particle identification.
3. Each particle is provided with the TOTA middleware enabling it to inject tuples across the network and to receive tuples sent by other particles. Tuples also enable each particle to know how many other particles are in its neighborhood (e.g. each periodically broadcasts "I am here" messages) to estimate the local density.

Particles do not have other capabilities other than the ones listed before. In particular:

4. They do not perceive the location (neither direction nor distance) of other particles, they do not have any kind of long range communication mechanism, nor a global accessible data space. In other words, although a particle can perceive how many particles are in the neighborhood, it can neither perceive in which direction and at which distance is a specific particle.
5. They do not have any notion of time and cannot rely on any global synchronization mechanism.

3.2 Morphogen Gradients

To let a swarm of mobile particles to self-organize into a global shape we took inspiration from biological morphogenesis. Morphogenesis is one of the major outstanding problems in the biological sciences. It concerns the fundamental question of how biological form and structure is generated starting from an immense number of biological cells. A mechanism common throughout embryo development and recognized as of primary importance towards morphogenesis is the use of *morphogen gradients* to determine positional information and polarity of cell. For instance, in the Drosophilae embryo, cells at one end of the embryo emit a morphogen (protein) that diffuses along the length of the embryo. The concentration of this morphogen is used by other undifferentiated cells to determine whether they lie in the head, thorax or abdominal regions. Different morphogens are used to determine the dorsal-ventral axis, wing development, and even leg bristle polarity.

Reproducing morphogen gradients in a network of short-range wirelessly interacting particles provided with the TOTA middleware is dramatically simple. A "source" particle can inject in the network a TOTA tuple that propagates across the network by increasing its content hop-by-hop. In the following, we will use the term *morphogen gradients* or simply *gradients* for this kind of tuple. The above very simple tuple can be used in powerful ways to influence the local activities of particles:

1. *Leader Election:* Gradients can be used to elect leaders in the group. Randomly elected leaders could propagate 'leader' gradients through the network inhibiting others to become leaders on their turn [12].
2. *Region Selection:* TOTA tuples can also specify in their propagation rule the maximum number of hops they are allowed to travel. If a leader particle propagates a gradient of this kind, it can create (approximately circular) regions of controlled size, and have other cells recognize their being in a specific circular region (by reading the gradient value). We outline that if the gradient value is incremented by one at each step, it provides an estimate of distance from the source: a perceived value of *n* steps implies a distance *nr* from the source, where *r* is the wireless communication range of particles. The quality of this estimate depends on the density of particles [13].
3. *Coordinate System:* Gradients can be used to self-organize coordinate systems. Particles, in fact, can recursively evaluate their coordinates by triangulating the distances – expressed by means of gradients – from elected beacons [13].
4. *Communication:* Gradient can be used to broadcast messages to other particles and, by having these messages follow other gradients previously laid down, effective routing mechanism can be enforced [11,14].
5. *Driving Motion:* if a particle can perceive the local slope of any gradient, it can also move following gradients uphill, downhill or along equipotential lines [2, 10, 12]. Moreover, since gradients are automatically update to reflect network movements an eventual driving direction will remain consistent despite network topology changes. With this regard, it is worth noting that since particles do not perceive other particles location, they do not perceive in which direction a perceived morphogen gradient decreases. To follow it downhill, a particle has to randomly wander until it perceives the gradient is going in the correct direction (i.e., the gradient it was following downhill is now perceived with decreased value).

From a methodological viewpoint, particles exploit the TOTA middleware and TOTA tuples to self-organize their respective positions in space. In particular, starting from any spatial configuration of particles:

1. particles can inject a TOTA tuple implementing some morphogen gradients;
2. particles react to locally perceived TOTA tuples by trying to follow gradients downhill/uphill (in a random way, as specified in the point 5 above), or by changing their activity state possibly depending on the perceived value of the tuples;
3. changes in the activity state of particles can lead to inhibiting the propagation of some TOTA tuples and/or to the diffusion of new types of tuples in the system,

leading back to point 1. One can then apply this process several times, with new types of tuples being propagated in different phases, so as to incrementally have particles self-organize into the required shape.

4 Experiments

Here we present a set of experiments of pattern formation exploiting the morphogen gradients approach. The experiments have been performed on the TOTA emulator already introduced. In the experiments, we assumed that particles have a certain physical size and cannot cross through each other.

4.1 Barycenter

In this example, starting from a random distribution of particles, a sort of distributed leader election algorithm is executed to identify the particle closest to the barycenter, i.e., the "center of gravity", of the whole system. Specifically, given *n* particles, the barycenter is that particle that minimizes the sum of the distances from the *n* particles. We remark that "leader" means a particle that has differentiated its behavior on the basis of its local properties or its perceived gradients and not because it has particular capabilities.

Detecting the barycenter of the system is very important for pattern formation, in that it identifies a reasonable point to refer to start subsequent shape formation activities. Morphogen gradients, expressing the approximate distance from their respective sources, enable the definition of a simple algorithm for the identification of the barycenter.

The algorithm: Each and every particle propagates a BARYCENTER gradient whose value increases by one at each step. Each particle senses BARYCENTER gradients propagated by all the other particles as they arrive, and adds their values together, call the resulting value *totGradients*. *totGradients* is the sum of distances from all the other particles. Therefore, the particle having the minimum *totGradients* is the barycenter. Since *totGradients* decreases monotonically to the barycenter, each particle can understand whether it has *totGradients* minimum or not, by simply comparing its value with the neighbors' ones. If no neighbors have a lower value of *totGradients*, the particle is the barycenter.

We emphasize the algorithm does have a well-defined termination point. Simply, each particle keeps on waiting for the income of new BARYCENTER gradients, to evaluate over and over whether it is the barycenter or not. Eventually, the algorithm converges, and particles will no longer receive any new gradient. The evolution of a sample simulation of the barycenter election algorithms is reported in Figure 2. In this figure (as well as in following simulation figures). It can be noted that, during the process, some particles may temporarily recognize themselves as barycenter. However, eventually, a single barycenter remains.

Slight modifications to the algorithm can be defined to elect two particles aligned around the barycenter at a specific distance from each other, as well as to identify particles on the border of the structure.

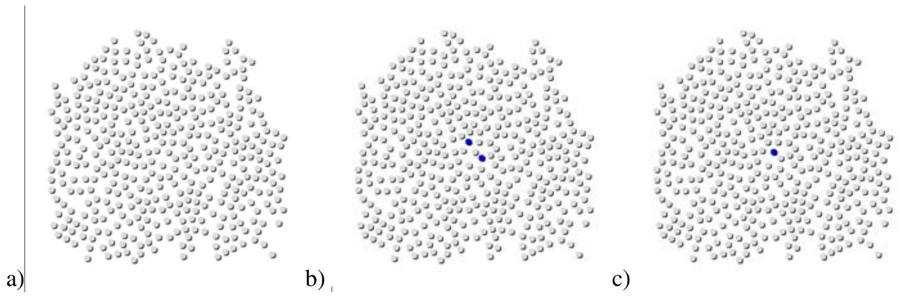


Fig. 2. Different stages of the establishment of the barycenter in a cloud of randomly distributed particles. As the system evolves, some particles (in black, rounded) may temporarily consider themselves the barycenter. Eventually, a single barycenter is left

```
  // sum of distance evaluated up to now
  totCount = 0
  // number of gradients received
  totGrad = 0
  // inject tuples to rend variables
  // visible to neighbors
  tota.inject(new Tuple("count", totCount))
  tota.inject(new Tuple("tot", totGrad))
  // inject distance tuple
  tota.inject(new Gradient(uniqueNumber))
  readLabel:
  Vector readV = tota.read(new Gradient())
  if readV.size() != totGrad
       totGrad = readV.size()
       tota.inject(new Tuple("tot", totGrad))
       goto readLabel;
  else
       // read other totGrad
       Tuple otherTotGrad = new Tuple("tot")
       Vector ot = tota.readOneHop(othetTotGrad)
       for every i in ot
               if totGradient!=ot[i].totGrad
               goto readLabel;
               end if
       end for
  end if
  // Have received all the gradients
  totCount = sumOfGradientsValue(readV)
  for every node in neighbor
       if neighborTotCount < totCount
       return NOT_BARYCENTER
       end if
  end for
 return BARYCENTER
```

Fig. 3. Pseudo-code of the barycenter algorithm

In the followings, to ground the discussion, we will provide a pseudo-code implementation of the discussed algorithms. Pseudo-code has been chosen to avoid the verbosity of real code and converting it in java should be straightforward. In the pseudo-code we put in boldface the instructions accessing to the TOTA middleware to highlight its role (see figure 3).

4.2 Circle

In this example particles run a distributed algorithm to assume a circle shape.

The Algorithm: Each particle runs the barycenter algorithm described in the previous section. The resulting barycenter particle will serve as the circle center. This particle propagates a CIRCLE gradient which increases its value by one at each step. All the other particles sense the gradient. If they sense a value greater than R (intended circle radius) they move along the decreasing propagation direction of the CIRCLE gradient. Eventually, all particles outside the intended circle radius will collapse toward it.

We want to remark that, as stated in Section 2, our particles cannot sense in which directions a gradient decreases. Therefore, a particle have to randomly chose a direction to move and, if the particle senses that the gradient of interest does not decrease – wrong guess – it can simple invert its direction. The key drawback of this technique is that it makes it possible for some particles to get lost, i.e., get disconnected from the network without any further information about where the rest of the particles are. However, since these unlucky events are extremely rare, and since individual particles are not important, this causes no harm.

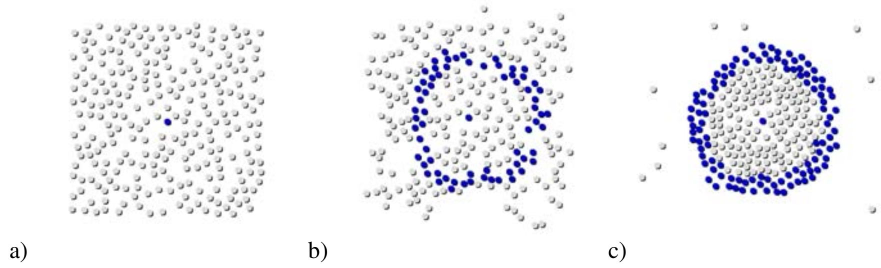


Fig. 4. Different stages of the circle formation. As the barycenter starts propagating the circle gradient, some particles (in black) already recognize themselves as being at the correct distance from the center and do not move; the other particles gradually collapse toward the circle circumference

```
if particle == BARYCENTER
    tota.inject(new Gradient("CIRCLE"))
end if
while(1)
    Vector readV = tota.read(new Gradient("CIRCLE"))
    Tuple circle = readV[0];
    if circle.value > R
        moveDownhill(circle)
    end if
end while
```

Fig. 5. Pseudo-code of the circle algorithm

The pseudo code of the circle algorithm is in Figure 5. Also in this case, the algorithm does not end: simply, the particles that found themselves inside the circle will stop moving. The result of a sample simulation is in Figure 4.

It is important to note that, despite the fact that during the execution of the barycenter algorithm some particles may temporarily consider themselves the barycenter, this causes not harm to the circle algorithm. Simply, these particles will temporarily diffuse a spurious CIRCLE gradient.

As an additional note, we emphasize that the execution of the circle algorithm in the presence of two barycenter enables the forming of elliptic shapes.

4.3 Ring

In this example, particles run a distributed algorithm, simply extending the circle one, to cooperatively assume a ring shape.

The Algorithm. Once the barycenter algorithm has run, and the CIRCLE gradient has been propagated, the particles on the circumference (i.e. those perceiving the CIRCLE gradient with value R) start propagating a RING gradient which increases its value by one at each step. This RING gradient, which also propagates to the inner parts of the circles, attracts particles towards the circumference, thus emptying the inside of the ring. The thickness of the ring can be tuned by having particles stop following the RING gradient when it reaches a value of T, where T will consequently be the thickness.

The pseudo code of the ring algorithm is in Figure 7. The result of a sample simulation is in Figure 6.

As for the case of the circle, we outline that by executing the ring algorithm in the presence of two barycenter "8"-like shapes can be obtained.

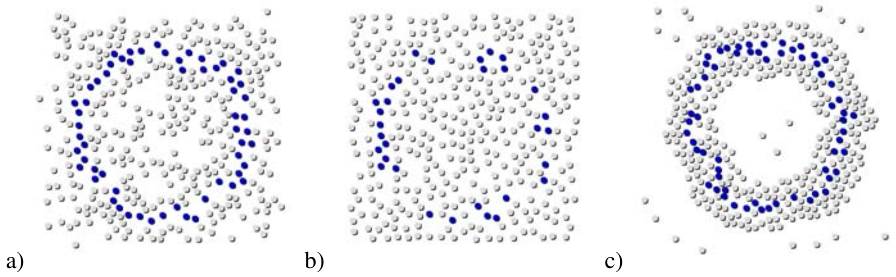


Fig. 6. Different stages of the ring formation. As the particles at the correct distance from the barycenter self-recognize to be there, they start injecting a gradient that attracts inner particles

4.4 Making Lobes

In this experiment, we tried to break the circular symmetry of previous experiments and let lobes emerge in the global shape. The overall idea is to exploit particles density as the source to break the symmetry. For instance, when forming a circle, particles start to collapse toward the circle itself. If the number of particles compared to the circle size is very high, then the perimeter of the circle will be very crowded.

```
if particle == BARYCENTER
    tota.inject(new Gradient("CIRCLE"))
end if
Vector readV = tota.read(new Gradient("CIRCLE"))
Tuple circle = readV[0];
if circle.value == R
    tota.inject(new Gradient("RING"))
else
     readV = tota.read(new Gradient("RING"))
     Tuple ring = readV[0];
     while ring.value >= T
        moveDownhill(ring)
     end while
end if
```

Fig. 7. Pseudo-code of the ring algorithm

Our idea is to force particles in very crowded areas to rearrange their positions so as to stay more separate form each other (remember that our particles can sense how many other particles are in the neighborhood). This process ends up in a slight deformation of the circle (i.e., in the emergence of small "lobes") in those part of its circumference where an excess of particles are accumulating. This small emergent lobes can be amplified via an additional mechanism of morphogen gradient inhibition that, in turn, makes larger lobes emerge.

To this end, and with reference to the circle, it is worth noting that the emergence of the circle shape directly derives from having the CIRCLE gradient spread in every direction uniformly. In this way, all the particles sensing the value R of the CIRCLE gradient end up in being almost equidistant form the center or, when the density is taken into account, in the circumference of a circle with small lobes However, if one makes the CIRCLE gradient increase its value slower in zones of high density then, in these zones, the gradient would reach the value R farther from the source. Particles, following that gradient, would not dispose on a circle, but on a circle with a lobe, where the lobe would correspond to the place in which the gradient reaches value R farther.

The Algorithm. Particles runs the CIRCLE algorithm and, upon receiving the CIRCLE gradient, have to re-propagate it. However, before doing that, particles sense the number of other particles in their neighborhood. If the number of particles in the neighborhood exceeds a specified threshold (*criticalDensity1*), the particle sets the rate at which the field increases to 0. This increasing rate will be reset to the default value of one when the density falls below another specified threshold (*criticalDensity2*).

The pseudo code of this algorithm is in Figure 9. The result of a sample experiment is in Figure 8. We outline that the algorithm does not enable to predict where in the circle lobes will form, and how may lobes will eventually form. This is an emergent characteristic of the system, that critically depends on two non-controllable factors: the initial disposition of particles and the outcome of the random movement of particles towards the center.

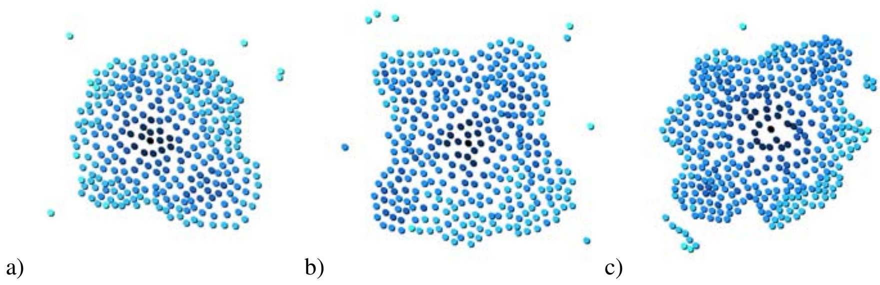


Fig. 8. Different circles with lobes. The particles in the circle that detect a high density of particles, inhibit the propagation of the circle gradient, thus leading to the formation of lobes

```
if particle == BARYCENTER
    tota.inject(new Gradient("CIRCLE"))
end if
Vector readV = tota.read(new Gradient("CIRCLE"))
Tuple circle = readV[0];
if circle.value > R
    moveDownhill(circle)
else if circle.value == R
    // making particles escape from crowd make
    // small lobes emerge
    moveAwayFromCrowd()
end if
// the following, makes even large lobes emerge
if numNeighbors > criticalDensity1
    // set the gradient increasing rate to 0
    circle.setAddValue(0)
end if
if numNeighbors < criticalDensity2
    // restore default increasing rate
    circle.setAddValue(1)
end if
```

Fig. 9. Pseudo-code of the lobes algorithm. To move away from crowd a particle chooses a random direction and follows it if it senses that the number of neighbors is decreasing

4.5 Polygon

The emergent phenomena of lobes in the previous section is interesting. Still, it would be important to have a way of controlling such emergent behaviors. In a further set of experiments we tried to enrich the algorithm to control the number of lobes to be created so as to obtain regular polygon shapes (e.g. triangles exagons, etc.).

The idea to control the number of lobes is again rooted in a leader election mechanism. We want to design an algorithm to elect *n* leaders on the circumference of the circle. These leaders must be equidistant from one another. Once this has been accomplished, the leaders will be in charge of adopting the trick described in the previous section, i.e., avoiding to increase the value of the CIRCLE gradient being re-propagate, so as to force the emergence of *n* lobes equidistant from one another and, consequently, leading to a nearly regular *n*-polygon shape.

The Algorithm. (i) each node runs the circle algorithm, *(ii)* once a particle on the circumference of the forming circle recognize their being in the current position, it starts casting random numbers; *(iii)* each node casting a number greater than a specified threshold becomes a leader – the threshold (T) is chosen so that it is very unlikely that two nodes become leaders shortly one after another; *(iv)* the leader starts propagating an ELECT gradient, that propagates only in the circle perimeter region (i.e., particles that are not on the ring inhibit its propagation); *(v)* nodes receiving the ELECT gradient, stop casting random numbers and if the received gradient value overcomes another specified threshold *L,* they become leaders on their turn; *(vi)* each leader sets the ELECT gradient value to 0 and continues its propagation; *(vii)* once that ELECT gradient is fully propagated there should be almost *(circle-length)/L* equidistant leaders on the circle. Thus *L* is a parameter controlling which polygon will emerge.

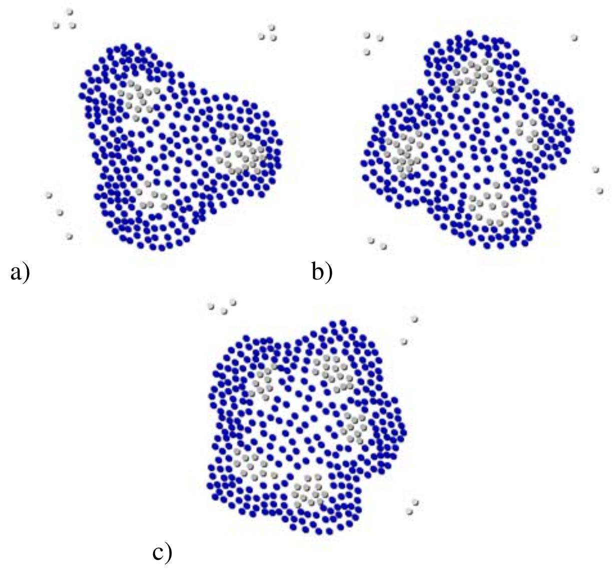


Fig. 10. Different polygon shapes obtained by multiple lobes: from triangle to pentagon

4.5 Self-repair

Whatever the shape being formed, it is of fundamental importance to preserve and maintain it despite particles being destroyed or added to the system. Such kind of maintenance cannot be performed from the external, particles themselves must be able to self-recognize a change in the configuration and self-repair their configuration. One of the key point in our approach is to enable such kind of self-repairing activities.

As shown in the previous experiments, the propagation pattern of the gradient is responsible of how the particles will dispose around the sources of that gradient. In any case, the gradient exerts a continuous attractive force on the particles, which perceive this force even when they are packed in the final shape and any movement is

```
if particle == BARYCENTER
    tota.inject(new Gradient("CIRCLE"))
end if
Vector readV = tota.read(new Gradient("CIRCLE"))
Tuple circle = readV[0];
if circle.value == R
readV = tota.read(new Gradient("ELECT"))
while readV.size() == 0
     if nextRandom() > T
     iAmLeader();
     tota.inject(new Gradient("ELECT"))
     circle.setAddValue(0)
end while
Tuple elect = readV[0];
if elct.value > L
     iAmLeader();
     tota.inject(new Gradient("ELECT"))
     circle.setAddValue(0)
end if
if circle.value > R &&  elect.value == 1
    circle.setAddValue(0)
end if
while(1)
    if circle.value > R
        moveDownhill(circle)
    else if circle.value == R
        moveAwatFromCrowd ()
    end if
end while
```

Fig. 11. Pseudo-code of the polygons algorithm

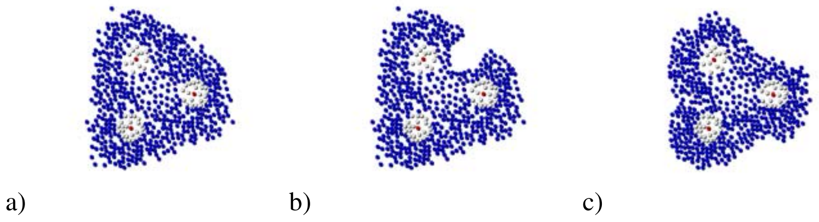


Fig. 12. Self-repair in a polygon shape. In (b) a number of particles is removed from the ensemble. After a short-period of time (c) the configuration is reconstructed

impossible (recall that particles cannot cross each other). The continuous attractive force and the fact that particles have a volume make our system able to self-recognize a change and self-repairing. For example, suppose that the shape is correctly created and in a certain zone some particles get destroyed for an impact, so opening a void space. On the one hand, particles are able to estimate their local density, and thus they can sense a sudden drop in their neighborhood revealing a change (see figure 12-b). On the other hand, all the particles close to the space previously occupied by destroyed particles now have the possibility to move, and because they are always subjected to the attractive force exerted by the gradient, they find a way to follow downhill the force field (see figure 12).

5 Related Work

In the last few years, several approaches targeting control algorithms for multi-robot systems have been proposed, which address goals similar to ours.

Several proposals in the area define distributed algorithms for pattern formation in robots exploiting the strong assumption that each robot, via visual observation, can determine the positions and movements of all other robots [7, 19]. This hypothesis makes it rather trivial to promote the formation of a variety of global patterns but it is hardly applicable to micro- or nano-robots/particles. Issues of scalability, battery consumption, line-of-sight, cost of global localization, etc. all call for a strictly local perception of the environment.

Other approaches have been proposed requiring robots witch strictly local perception, but still requiring them the capability to detect the distance and the direction of neighbor robots[1, 5]. The key idea is that robots, by positioning themselves at specific distances and directions from other robots, can self-organize in a variety of regular shapes. Little is said about the possibility of making more complex shapes emerge, e.g., by making information flow from robot to robot (as in the case of morphogen gradient).

A possible way to promote the formation of spatial patterns in the absence of distance and direction information is to get inspiration from the way chemicals and crystals grow into self-organized regular structures. Approaches of this kind are explored, for instance, in [8, 9, 17]. The general idea (with specific differences characterizing different proposals) is to exploit stateful particles capable only of sensing the internal state and the presence of other particles (either via proximity sensing or via direct contact). Particles are deployed together in an environment and there start randomly moving. When particles keep in touch with each other, they apply internal transition rules (based on their own state and on the state of close particles) to decide whether to "stick" to that position or continue moving. Unfortunately, the approach enables the direct programming of transition rules leading to very simple and regular patterns only. More complex patterns require complex search heuristics to determine a set of transition rules leading to the desired global pattern. Also, the process leads to the formation of static non-adaptive patterns.

Algorithms for the control of shape and motion in modular robot have been proposed exploiting an approach strictly related to ours [16, 18]. There, "hormones" (similar to morphogen gradients) are created and propagated through the modules of the robot. Robots' modules decide how to move on the basis of a lookup table associating their local configuration and the locally perceived hormones with the next action. The result is to have the robot modules self-organize into globally coherent shapes or into globally coherent motion patterns (i.e. gait). Although some of the result of this approach are excellent, one should consider that such representation must be locally stored by each robot, which may not be possibly for robot with very limited resources. Moreover, relying on an a priori representation of the shape, robots deployed in a constrained environment where the shape does not fit would be left with no choices but leaving the shape incomplete, rather than flexibly adapt the shape.

The approach proposed in the Amorphous Computing project [12] for the formation of origami shapes in a amorphous network of particles is the one that most directly relates to our work. Particles, by communicating only with their local neighborhood, can self-organize into various patterns of activity by propagating morphogen gradients and changing their state according to the perceived morphogen gradients. The main difference from our work is that here particles can't move altering their respective topologies. Furthermore, some particles (i.e., those at the border) must be started in a special initial state, thus requiring an a priori differentiation.

6 Conclusions and Open Directions

The TOTA approach enables the effective formation of a variety of complex shapes in computational particles with minimal capabilities.

Despite these promising results, a number of open directions are still to be investigated. Currently our group is interested to: experiencing differentiation of activities and global coordination based on cellular automata inspired approach; trying to define a simple and modular programming model; trying to achieve – other than the formation of static patterns – coherent motion gaits in particles; building some hardware prototype for particles and validating our approach in the real world.

References

[1] J. S. Bay, C. Unsal, “Spatial Self-Organization in Large Populations of Mobile Robots.”, IEEE Symposium on Intelligent Control, Columbus (OH), 1994.

[2] D. Coore, “Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer”, PhD Thesis, MIT, 1999.

[3] S.J. Day, P.A. Lawrence, “Morphogens: Measuring Dimensions: the Regulation of Size and Shape”, Development 127:2977-2987, 2000.

[4] D. Estrin, D. Culler, K. Pister, G. Sukjatme, “Connecting the Physical World with Pervasive Networks”, IEEE Pervasive Computing, 1(1):59-69, 2002.

[5] J. Fredslund, M. Mataric, “A General Algorithm for Robot Formations Using Local Sensing and Minimal Communication.”, IEEE Transactions on Robotics and Automation, 18(5):837-846, 2002.

[6] D. Gelernter, N.Carriero “Coordination Languages and Their Significance”, Communication of the ACM, Vol. 35, No. 2, pp. 96-107, 1992.

[7] N. Gordon, I.A. Wagner, A.M. Bruckstein, “Bee Dance Algorithm for Pattern Formation on a Grid.”, IEEE Conference on Intelligents Agents Technologies, Toronto (CA), 2003.

[8] Y. Guo, G. Poulton, P. Valencia and G. James, “Designing Self-Assembly for 2-Dimensional Building Blocks”, in Engineering Self-Organizing Applications, LNCS No. 1977, Springer Verlag, 2004.

[9] C. Jones, M. Mataric, “From Local to Global Behavior in Intelligent Self-Assembly”, IEEE Conference on Robotics and Automation, Taipe (TW), 2003.

[10] M. Mamei, F. Zambonelli, L. Leonardi “Co-Fields: a Physically Inspired Approach to Distributed Motion Coordination”, IEEE Pervasive Computing, 3(2):52-61, 2004.

[11] M. Mamei, F. Zambonelli, “Programming Pervasive and Mobile Computing Applications with the TOTA Middleware”, IEEE Percom, Orlando (FL), 2004.

[12] R. Nagpal, “Programmable Self-Assembly Using Biologically-Inspired Multirobot Control”, ACM Joint Conference on Autonomous Agents and Multiagent Systems, Bologna (I), 2002.
[13] R. Nagpal, H. Shrobe, J. Bachrach, “Organizing a Global Coordinate System from Local Information on an Ad Hoc Sensor Network”, in Information Processing in Sensor Networks, LNCS No. 2643, Springer Verlag, 2003.
[14] R. Poor, “Embedded Networks: Pervasive, Low-Power, Wireless Connectivity”, PhD Thesis, MIT, 2001.
[15] M. Shang, W. Ruml, Y. Zhang, “Localization from Mere Connectivity”, ACM Conference on Mobile ad-hoc Computing and Networking, Annapolis (MD), 2003.
[16] W.M. Shen, B. Salemi, P. Will, “Hormone-Inspired Adaptive Communication for Self-Reconfigurable Robots”, IEEE Trans. on Robotics and Automation, 18(5):1-12, 2002.
[17] W.M. Spears, D.F. Gordon, “Using Artificial Physics to Control Robots.”, IEEE International Conference on Information, Intelligence and Systems, 1999.
[18] K. Stoy, R. Nagpal, “Self-Reconfiguration Using Directed Growth, 7th Int. Symposium on Distributed Autonomous Robotic Systems”, Toulouse (F), 2004
[19] K. Sugihara, I. Suzuki “Distributed Motion Coordination of Multiple Mobile Robots.”, IEEE Int'l Symp. on Intelligent Control, Philadelphia (PN), 1990.

Directed Self-assembly of 2-Dimensional Mesoblocks Using Top-Down/Bottom-Up Design

Geoff Poulton, Ying Guo, Geoff James, Phil Valencia,
Vadim Gerasimov, and Jiaming Li

CSIRO ICT Centre,
Marsfield NSW 2122, Australia
{Geoff.Poulton, Ying.Guo, Geoff.James, Phil.Valencia,
Vadim.Gerasimov, Jiaming.Li}@csiro.au

Abstract. In this paper we present a general design methodology suitable for a class of complex multi-agent systems which are capable of self-assembly. Our methodology is based on a top-down, bottom-up approach, which has the potential to achieve a range of global design goals whilst retaining emergent behaviour somewhere in the system, and thereby allowing access to a richer solution space. Our experimental environment is a software system to model 2-dimensional self-assembly of groups of autonomous agents, where agents are defined as square smart blocks. The general design goal for such systems is to direct the self-assembly process to produce a specified structure. The potential of this design methodology has been realised by demonstrating its application to a "toy" problem – the self-assembly of rectangles of different sizes and shapes in a two-dimensional mesoblock environment. The design procedure shows different choices available for decomposing a system goal into subsidiary goals, as well as the steps needed to ensure a match to what is achievable from the bottom-up process. Encouraging results have been obtained, which allows mesoblock rectangles of specified size to be assembled in a directed fashion. Two different approaches to the same problem were presented, showing the flexibility of the method.

1 Introduction

The study of complex systems is now over two decades old and much has been discovered about such systems, whose best-known characteristic is the emergence of self-organised global properties from the aggregate behaviour of its constituent components [2]. A good definition of self-organisation is given by Hermann Haken [1]: *a system is self-organizing if it acquires a spatial, temporal or functional structure without specific interference from the outside. By "specific" we mean that the structure or functioning is not impressed on the system, but that the system is acted upon from the outside in an non-specific fashion.*

Most complex systems research has employed the traditional scientific method of hypothesis and test, a "bottom-up" approach which works from fundamental knowledge and experimentation to derive new principles about the world. However, the very property which characterises complex systems also works to make difficult the formulation of general theories - the inherent unpredictability of emergent behaviour. There is no reason to suppose that this situation will change in the near future.

In contrast, engineering design is a “top-down” process which starts with a global goal and seeks methods to achieve it using available scientific knowledge. For complex systems this has proven to be difficult for reasons stated above - general scientific principles are hard to come by.

There are two approaches to the design of complex systems. The first is to use evolutionary computation or similar methods to obtain designs for specific instances [8,9]. However, such designs rarely generalise to allow generic design principles, again because of the unpredictability of the underlying system. Secondly, the system itself may be modified or restricted to remove emergence, allowing traditional design methods to apply [10]. The disadvantage is that, even when it is possible, solutions will be inferior because of the loss of the rich solution space provided by complexity, and which biological systems use to such advantage.

In this paper we present a design approach which is a compromise between these two extremes, by working simultaneously from the top down and the bottom up in an attempt to define intermediate “entities” which retain emergence and yet allow the possibility of formulating broad design principles. This concept will be developed in the context of multi-agent systems for which the principal self-organising behaviour is self-assembly. The general design goal for such systems is to direct the self-assembly process to produce a specified structure.

More specifically, the agents in our system will be simulated 2-dimensional square “mesoblocks”, in analogy to the physical mesoblocks investigated by Whitesides and others [5,6], but imbued with a rudimentary intelligence. Large numbers of identical blocks, moving randomly in a “sea”, interact according to their internal properties whenever two blocks come together [3,4]. The surfaces are polarised as negative (-1), positive (+1) or neutral (0), so that opposite polarities will attract and stick together. No other combination of polarities will allow blocks to stick [3,4]. While a large number of objects can be constructed from these static agents by varying their surface properties, a richer variety of self-assembled objects is possible by allowing edge polarities to change following an “event”, under the control of an internal state machine. In this study an event is defined as a block either sticking to or unsticking from another block (although other choices are possible). For simplicity we assume all “sea” blocks to have the same fixed physical shape and internal state machine.

An important concept from our earlier work is the “enzyme”, which allows a more flexible and directed method of producing basic building objects [4]. In the real world an enzyme is an organic protein which catalyses a specific reaction. Here, we use it to characterize an assembly of blocks which is capable of producing another block assembly whilst itself remaining unchanged. By introducing an enzyme into the multi-agent environment, the structures which self-assemble depend also on the rule set and physical structure of the enzyme. Different enzymes will generate different final objects from the same “sea” blocks. Enzymes can also give control over where and when structures self-assemble, by appropriate placement and assuming that enzymes can be switched between active and inactive states. The properties of agents and enzymes are fundamental to the directed self-assembly process.

The remainder of this paper is organized as follows. Section 2 outlines our approach, Section 3 describes the 2-dimensional self-assembly environment and enzymes. In Section 4, we present two examples of the top-down/bottom-up design process in a simulation environment. Finally, conclusions based on these experiments are discussed in Section 5.

2 Top-Down/Bottom-Up (TDBU) Design

Our approach to creating a general framework for the design of complex multi-agent systems is to seek a balance between "top-down" (engineering) and "bottom-up" (scientific) processes. Engineering design starts with a system goal and employs a top-down approach to formulate more achievable intermediate goals. In contrast, the scientific method develops new knowledge of what is achievable by working from the bottom-up. Successful design is possible when the two processes can be matched, with intermediate engineering goals being capable of being achieved using existing scientific understanding. The particular problem with complex systems in general, and multi-agent systems in particular, is the difficulty of formulating general scientific principles describing system behaviour.

It may often be possible, using a hierarchical design process, to break a range of system goals into sub-goals which are within the capacity of scientific knowledge. Solving the sub-goals would then allow any system goal within the range to be achieved. The danger in doing this is that the emergent behaviour characteristic of complex systems may be lost, denying the designer the rich solution space offered by such systems.

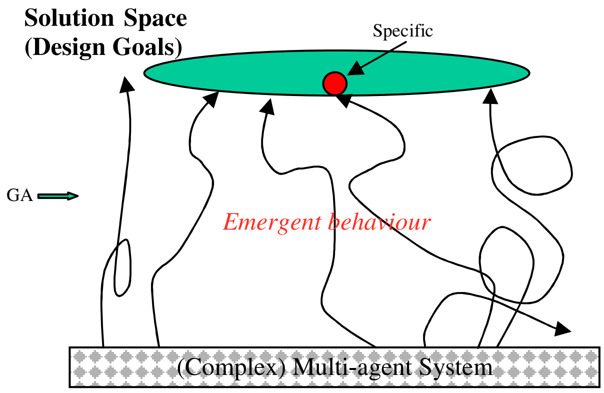


Fig. 1. Single-layer system design. The solution is specific for one problem, and it is difficult to find generic design rules

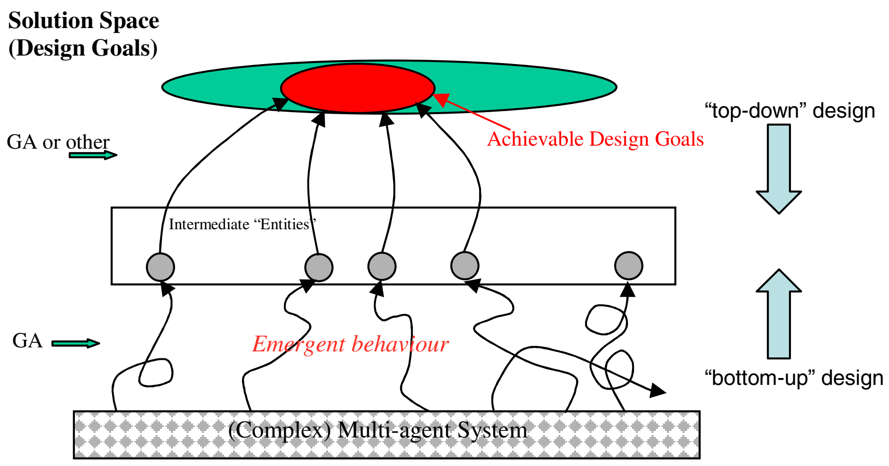


Fig. 2. Intermediate entities, capable of achieving a range of goals, are derived by a "top-down" process from the design goals. The "bottom-up" process is to evolve the intermediate entities from the multi-agent system. This leads to a broader solution space than Fig. 1

To access this rich space of potential solutions, it is important to preserve emergent behaviours that would be lost with a fully hierarchical, engineering design. Our approach, involving a minimal hierarchical decomposition, is a means of seeking a balance.

It is valid to ask whether any such decomposition is necessary, when many examples exist where complex systems are designed for a specific goal using evolutionary computation or similar methods [8,9]. The answer is that we aim to develop methods which will allow a range of design goals to be achieved without having to repeat time-intensive evolutionary computations. This point is explained in more detail as follows.

Fig. 1 illustrates a single-layer system where GA is used to achieve a given design in the solution space [3,4]. For systems exhibiting emergent behaviour such designs are not usually robust in the sense of allowing general rules for a class of designs to be formulated. The specific solution found may well be optimal, but may not generalise well because of the complex nature of the system, making it difficult to determine generic design rules. On the other hand Fig. 2 illustrates a TDBU process where we can retain emergence in one (or both) parts, thus broadening the solution space. This is possible because, although the "intermediate entities" may result from an emergent process, they can be used as generic building blocks to achieve a broader range of goals in the solution space. This will effectively give general design rules, at least within the broader area of achievable goals. Another advantage of the hierarchical design is that since the problem has been split, simpler optimization of the constituent parts will almost certainly result.

It should be noted that considerable choice may exist as to the set of intermediate entities in a given case. In part, this will be due to the fact that only a part of the whole solution space can be expected to be covered, so there will be some flexibility which depends on just which part is considered most important. Some solutions will be better than others; for example, a good solution would be if only a few entities could span a large part of the space.

2.1 TDBU Design for Mesoblock Self-assembly

Previous work by the authors on TDBU design of multi-agent systems has focused on bottom-up analysis, investigating the class of substructures that can be directed to self-assemble by specifying the internal agent properties [3,4]. The most important outcome of this research has been a class of "enzymes", simple multi-agent configurations capable of producing other simple structures with desired properties whilst themselves remaining unchanged. Significantly, many self-replicating enzymes were also found [4], with the interesting property of reproducing themselves in addition to creating target structures.

In this paper, we will focus more on the top-down part of the TDBU design process. After discussing the various choices available for decomposing a system goal into subsidiary goals, as well as the steps needed to ensure a match to what is achievable from the bottom-up process, we will illustrate the design procedure by carrying out and simulating the entire TDBU procedure for a preliminary "toy" problem. Very encouraging results have been obtained for this problem, which allows mesoblock rectangles of specified size to be assembled in a directed fashion. Results of the simulations will be given in Section 4.

To design a self-assembly system, we want to keep the emergent behaviour of the system. A number of research groups, including CSIRO, have studied self-assembly at the "mesoscale" (in the order of millimetres to centimetres) using physical "mesoblocks" with a range of shapes and construction and whose edge adhesion properties may be varied. Such mesoblocks self-assemble into a wide range of regular lattice-like arrangements, depending on their properties, and can be regarded as analogues of nano- or molecular-scale systems [5-7].

In previous work by the authors [3,4] such blocks are regarded as agents that interact due to their edge properties, which are not static, but may change under the influence of an internal state machine. The resultant self-assembled object (if one exists) is dependant on the agents' properties including size, edge polarities and strength of edge fields as well as the parameters of the internal state machine. This state machine changes the polarity of any number of its sides following the detection of an "event", which is usually the sticking to or unsticking from another block. In [3], restricting the analysis to two-dimensional rectangular blocks, Guo et. al. evolved block parameters and rule sets of the state machine which allow the self-assembly of desired basic structures suitable for use as primitive building blocks for the directed assembly of more complicated objects.

3 Building Blocks for Self-assembly

As mentioned above, the agents in the self-assembly system are two-dimensional rectangular blocks which may combine to form intermediate entities (the most important of which are enzymes) in a TDBU design. Both the agents and intermediate entities become building blocks for the final self-assembly of a desired structure. The main properties of agents and enzymes are described in detail in the following sections.

3.1 2D Mesoblock Agents

At the nanoscale, self-assembly of nanostructures can be controlled by using the attraction and repulsion properties of attached complementary groups. We have included similar properties in our agents with the aim of generating insights into the self-assembly processes of multi-agent systems of this type, with the hope that these insights may then be transferred to real world environments. Of course, it will be necessary to find nano- or molecular analogues with rudimentary intelligence. Several authors have suggested mechanisms which may allow such properties [11,12].

With this aim in mind and referring to Figure 3 the properties of our agents may be defined as follows:

(1) Each block has four edges, each of which can have positive (+1), negative (-1), or neutral (0) polarity. This is illustrated in Figure 3, where the four edges of the block are labeled for easy description. The edge state of an agent will be described as a vector of four trinary numbers: $Q = (a_1, a_2, a_3, a_4)$, where $a_i \in \{+1, -1, 0\}$. For instance, the edge state of the block in Figure 1 can be described as $Q = (+1, 0, 0, -1)$.
(2) The internal state machine can change the polarity of each edge as the result of an event.
(3) Events detectable by each block are the acts of "sticking" or "unsticking" at one or more of its edges.
(4) The internal state machine in each block contains rules linking edge polarity changes to "sticking" or "unsticking" events.
(5) The initial state of all the blocks in one environment is identical. This includes the edge polarities as well as any internal states.

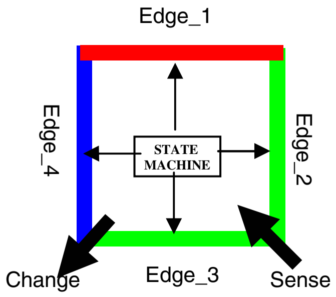


Fig. 3. Structure of each agent in the multi-agent system. Edge_1 is positive (+1), Edge_4 is negative (-1), and Edge_2, Edge_3 are neutral (0)

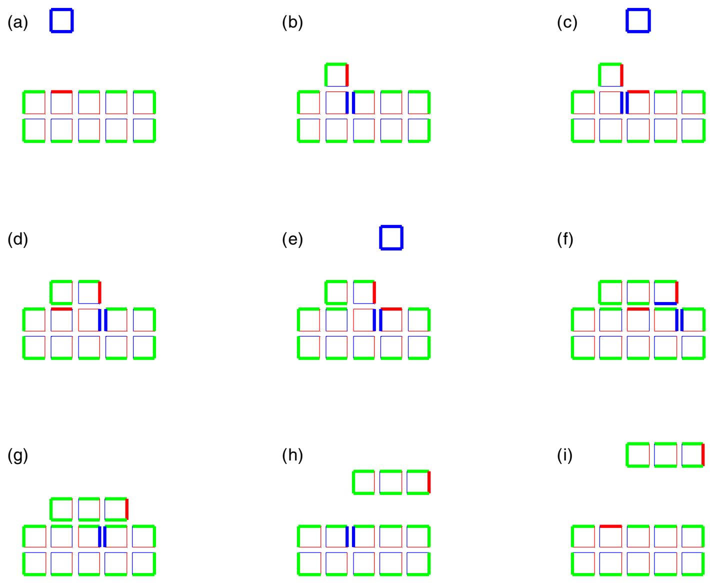


Fig. 4. A simple example of an enzyme. This is an artificially constructed 10-block structure sitting in a "sea" of blocks with all negative sides and an identical rule set. This enzyme produces linear groups of three blocks with all zero polarity except for one end, which is positive

In the simulation environment the blocks interact in the following manner:

(1) Edges of opposite polarity stick together and generate a "sticking" event which is passed to the internal state machines of both sticking blocks.
(2) Like polarities repel and will cause connected blocks to unstick. (This may also generate an event to be passed to the internal state machines, an option which is not used in this paper for reasons of simplicity).
(3) Neutral polarities will neither attract nor repel any other polarities. If a polarity changes to neutral between two connected blocks then they will unstick.
(4) Blocks move randomly in the 2-D environment under Brownian-like motion by steps which are multiples of the block width, with rotation of multiples of 90°.
(5) For a given situation many different structures may self-assemble. A block will separate from a structure if there is no net "sticking force" to hold it on. For the Q$^{\text{th}}$ block this may be determined from the separation function $A(\mathbf{Q}) = \sum_{i=1}^{4} p_i$, where p_i is the sticking force due to the i^{th} edge. p_i is +1 if the edge sticks to a neighbour, -1 if it is repelled and 0 if there is no neighbouring block or a neutral edge. Block $\mathbf{Q}$ will remain part of the structure if $A(\mathbf{Q}) > 0$, or will separate if $A(\mathbf{Q}) \leq 0$. Note that separation functions really should be defined for all structures, not just for single blocks. In general this is a difficult problem which has not been addressed in the present study.

3.2 Enzymes

The properties of an enzyme are defined as:

(1) Each enzyme is a stable structure comprising *K* blocks.
(2) Each block in the same enzyme has the same rule set.
(3) Blocks in the environment ("sea" blocks), whilst identical, can have a different rule set from the enzyme blocks.
(4) An enzyme must remain unchanged after the self-assembly process. During the process it may change in size and shape, but it must return finally to its initial state.

An example of an enzyme enabling a self-assembly process is shown in Figure 4. This enzyme may be simply modified to produce linear structures of any length.

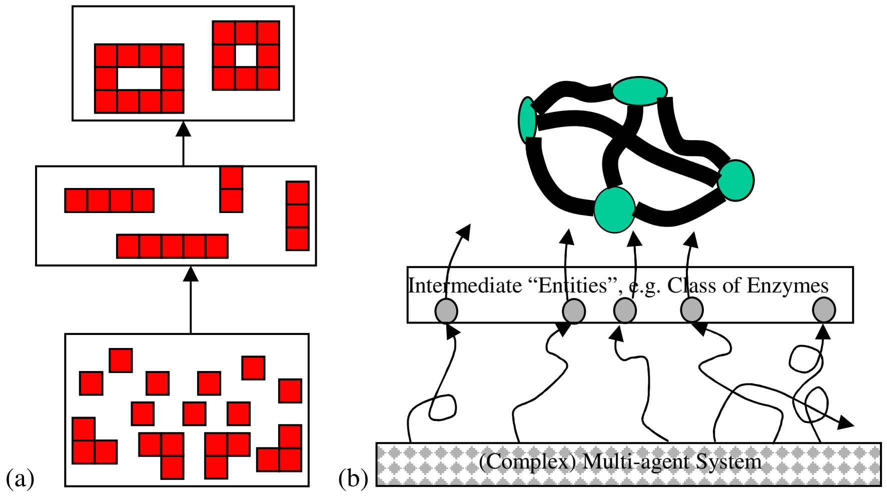

(a)
(b)
Intermediate "Entities", e.g. Class of Enzymes
(Complex) Multi-agent System

Fig. 5. Two examples of TDBU design. (a) self assembly of mesoblock rectangles. (b) self assembly of connecting mesoblock assemblies

4 TDBU Design of Self-assembly Systems: Experiments

In this section a simple example of a complete TDBU process will be selected and worked through, to illustrate the method and to bring out concepts and difficulties. For the two-dimensional mesoblock environment described above several such examples are possible. One which would be of value if it could be transferred to the nanoscale is self-assembling connectivity - the self assembly of mesoblock structures connecting a number of static structures placed randomly in the environment. This would have application to the self-assembly of nanoscale electronic circuits. Another, simpler example is the self assembly of mesoblock rectangles of controllable size and shape. Because this has no apparent nanoscale application it is really a "toy" problem. Nevertheless, it is selected for our experiment because of its simplicity. Fig. 5 shows a schematic of this experiment and also of self-assembling connectivity.

In what follows the minimal hierarchy TDBU design concept is developed with the abovementioned toy problem, whose aim is to design a class of rectangular structures of any given size or shape which self-assemble from two-dimensional mesoblocks. To illustrate the choices implicit in the design method we have employed two different design structures, both using enzymes but with different complexity. This illustrates the flexibility in the choice of intermediate entities inherent in the TDBU process. The two methods, together with results of simulation experiments, are described in the following sections.

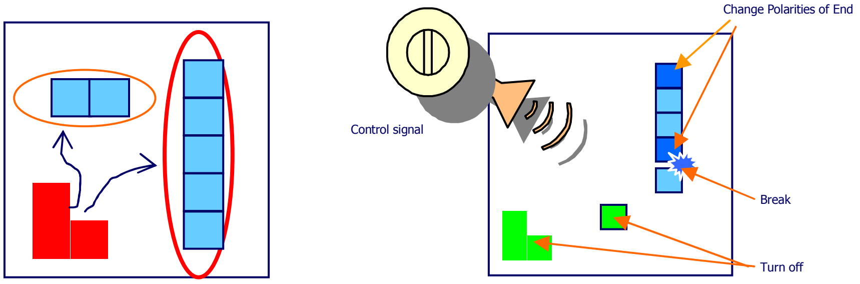


Fig. 6. Design process of approach one. The figure at the right shows the performances of the components when the control signal is broadcast to the whole environment

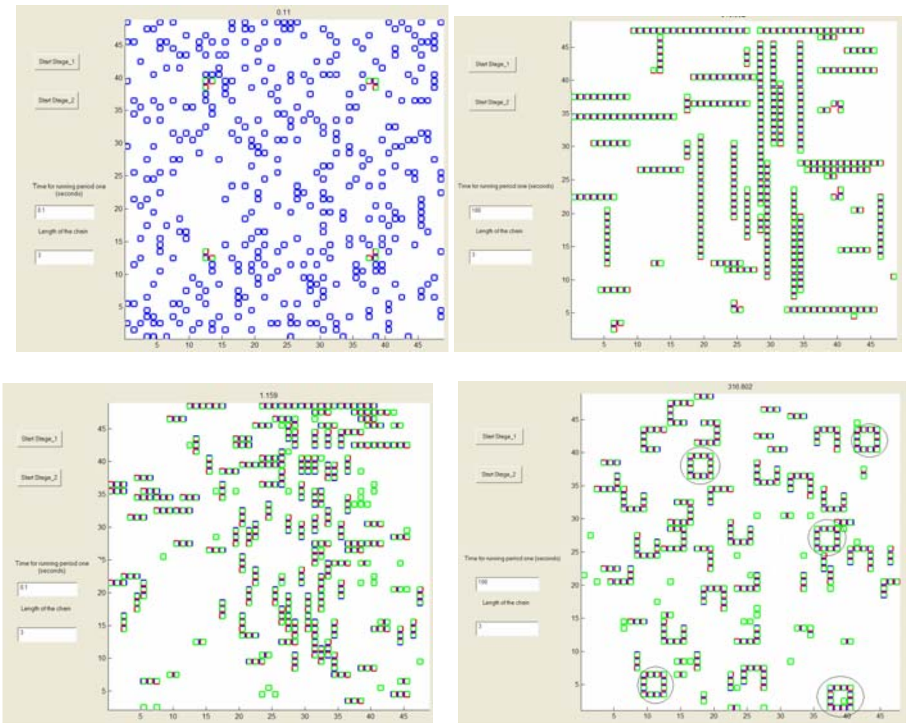


Fig. 7. Four different stages of the self-assembly environment based on a TDBU design. The self-assembled rectangles are circled in the last figure

4.1 Achieving the Goal with Fixed Enzymes

The first design approach is shown in Figure 6, which is a detail of the design process of Figure 5 (a). Two types of fixed enzymes, which produce horizontal and vertical strings, and the sea blocks are the basic building structures. This process of growing long chains will be allowed to continue for a while. Then "messages" are sent to the whole environment as control signals which impart the essential information about the goal --- in this case the length and breadth of the desired rectangles. Following receipt of the message all blocks, enzymes and chains in the whole environment will take the following actions, depending on their situation:

- All single blocks become neutral on all sides;
- All enzymes are deactivated;
- All long chains are truncated to the lengths given by the message, and the end block polarities are changed to allow them to self-assemble into rectangles.

Then all the chains with active ends (positive or negative) and the correct length will start to stick together and generate the rectangles. There are many possible methods of achieving this. One simple way is to include a counter in each block. During assembly by the enzyme the first block in the chain is set to 1, and the following blocks to 2, 3, etc. as the chain grows. When the global message is received, the blocks break with their next neighbours if their counter numbers divided by the given length are integers. This process also activates the ends of the chain.

To illustrate the operation of this process an artificial simulation environment was set up. Figure 7 shows four different states of this environment: (a) the initial condition, which includes four enzymes and many sea blocks. (b) the end of the first stage, with a number of chains of different lengths which self-assembled after activation by an enzyme based on their internal rule set. (c) the intermediate stage, following actions generated by the global control signal. (d) partway though the final self-assembly stage, with some rectangles generated (circled).

4.2 Achieving the Goal by Adjusting Enzymes

This design approach is shown in Figure 8, which is a detailed design process from the idea of Figure 5 (b), where the global goals were achieved by creating a flexible enzyme and then adjusting its properties according to the particular goal required. There are two large L-shape enzymes in this environment. These enzymes, and the rule sets of the sea blocks, are designed to ensure that particular L-shape structures are generated with particular polarities (+1 at one end). When these structures, moving freely in the environment, meet at their open ends, the positive (+1) will stick with negative (-1) polarity, and they will generate a rectangle whose dimensions are determined by the L-shaped structures produced by the enzymes. These dimensions are set by the positions of the "terminal" blocks in the enzymes, shown as the one with dot and arrow in Figure 8. Without external intervention only rectangles of a fixed size will be produced. However, as in the first example, external global messages may be used to shift the positions of the terminal blocks, and hence the size and shape of the rectangles produced. One way is to provide labels for all blocks in the enzyme allowing any of them to be activated by targeted messages, thereby becoming the new terminal blocks. Alternatively, the enzymes may be programmed to change this themselves after a fixed number of rectangles have been produced.

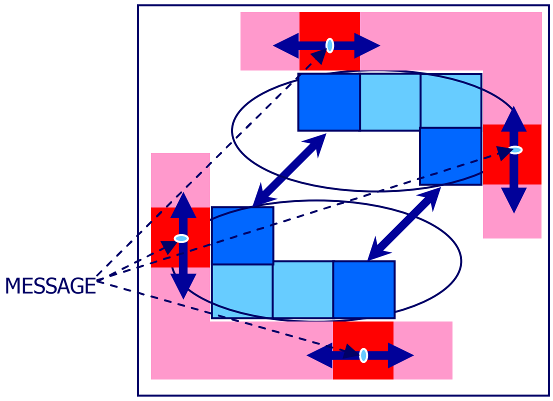


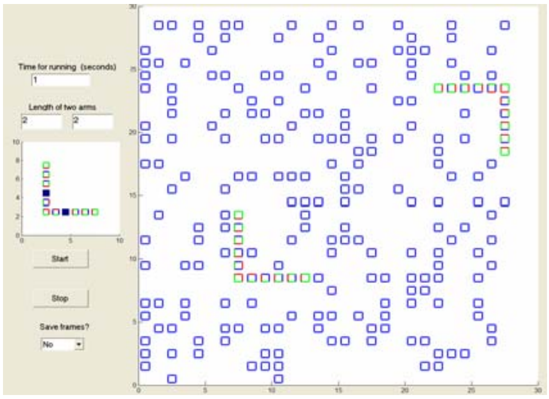
Fig. 8. Second design process using flexible enzymes

This design process can also be simulated in the same environment and this is illustrated in Figure 9. In the initial state (Figure 9.a), the two large L-shape structures are the enzymes in a "sea" of sea blocks. There is a small window in the left of the main environment, which shows the enzyme with the highlighted "terminal" blocks. Two different classes of rectangles generated by two different terminal block positions are shown in Figure 9.b and Figure 9.c.

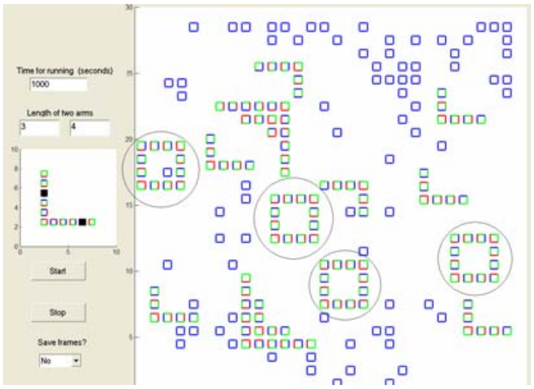
5 Conclusions and Further Directions

The essential idea of this paper is to present a general design methodology suitable for a class of complex multi-agent systems which are capable of self-assembly. This problem is difficult because such complex multi-agent systems generally exhibit emergent behaviour which makes global performance hard to predict, and hence very hard to design for. Our methodology is based on a top-down, bottom-up approach, which has the potential to achieve a range of global design goals whilst retaining emergent behaviour somewhere in the system, and thereby allowing access to a richer solution space. The potential of this design methodology has been realised by demonstrating its application to a "toy" problem – the self-assembly of rectangles of different sizes and shapes in a two-dimensional mesoblock environment. In fact, two different approaches to the same problem were presented, showing the flexibility of the method.

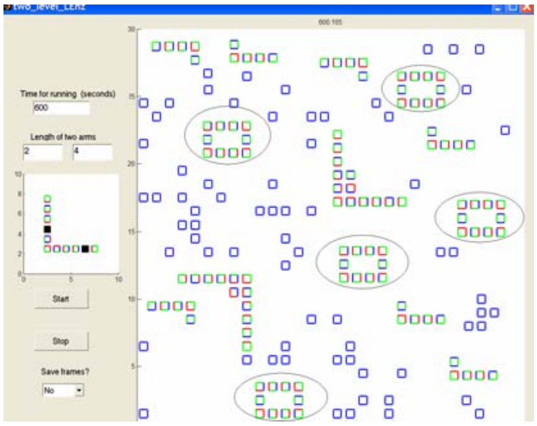
The next steps in this research will be to apply the technique to a wider range of problems within the 2-D mesoscale environment, concentrating on applications which have useful nanoscale analogues. Further work will be to change our environment to


Time for running (seconds): 1
Length of two arms: 2 2

(a)


Time for running (seconds): 1000
Length of two arms: 3 4

(b)


Time for running (seconds): 600
Length of two arms: 2 4

(c)

Fig. 9. TDBU rectangle design: achieving global goal by adjusting enzymes. (a) The initial state of the environment, with the enzyme shown in the small window. (b)The self-assembled rectangles of size 4 by 4 are circled. (c) The self-assembled rectangles of size 3 by 4 are circled

include a model of physics which is as close to the nanoscale as possible. Once this has been done we will be in a position to assess whether our approach for designed self-assembly would be feasible at the nanoscale.

References

1. Haken, H., Information and Self-Organization: A Macroscopic Approach to Complex Systems, Springer-Verlag, 1988.
2. Vemuri, V., Modeling of Complex Systems: an Introduction, New York, 1978.
3. Guo, Y., Poulton P., Valencia P., and James, G., Designing Self-Assembly for 2-Dimensional Building Blocks, ESOA'03, Melbourne, July 2003.
4. Poulton, G., Guo, Y., Valencia, P., James, G., Prokopenko, M., Wang, P., Designing Enzymes in a Multi-Agent System based on a GA, 8th Conf. Intell. Auton. Sys., 2004.
5. Whitesides, G., et al., Three-Dimensional Mesoscale Self-Assembly, J. Am. Chem. Soc., 1998, 120, 8267-8268.
6. Whitesides, G., et al., Design and Self-Assembly of Open, Regular, 3D Mesostructures, Science, 1999, 284, 948-951.
7. Raguse, B., Self-assembly of Hydrogel Mesoblocks, personal communication, CSIRO, 2002.
8. Surmann, H., Kanstein,A., Gosser,K., Self-Organizing and Genetic Algorithms for an Automatic Design of Fuzzy Control and Decision Systems, EUFIT 93 First European Congress on Fuzzy and Intelligent Technologies, Aachen 1993, Vol. 2, pp 1097-1104.
9. de Garis, H., Growing Arbitrary Shapes with Genetic Programming, Proceedings of the 7th International Conference on Machine Learning, 1990, pp.132-139.
10. Goldman,C., Zilberstein,S., Optimizing Information Exchange in Cooperative Multi-agent Systems, AAMAS 2003, Melbourne, Australia, 2003.
11. Winfree, E., Yang, X. & Seeman, N. C., Universal computation via self-assembly of DNA: Some theory and experiments. Proc. 2nd DIMACS Meeting on DNA-Based Computers, June 20–12, 1996.
12. Fennimore, A., Yuzvinsky, T., Wei-Qiang Han, Fuhrer M., Cumings, J., & Zettl, A., Rotational actuators based on carbon nanotubes, Nature, Vol. 424, 24 July, 2003, pp 408-410.

Analysis of a Stochastic Model of Adaptive Task Allocation in Robots

Aram Galstyan and Kristina Lerman

Information Sciences Institute,
University of Southern California,
Marina del Rey, California
{galstyan, lerman}@isi.edu

Abstract. Adaptation is an essential requirement for self–organizing multi–agent systems functioning in unknown dynamic environments. Adaptation allows agents to change their actions in response to environmental changes or actions of other agents in order to improve overall system performance, and remain robust even while a sizeable fraction of agents fails. In this paper we present and study a simple model of adaptation for task allocation problem in a multi–robot system. In our model robots have to choose between two types of task, and the goal is to achieve desired task division without any explicit communication between robots. Robots estimate the state of the environment from repeated local observations and decide what task to choose based on these observations. We model robots and observations as stochastic processes and study the dynamics of individual robots and the collective behavior. We validate our analysis with numerical simulations.

1 Introduction

Adaptation is an essential requirement for multi–agent systems functioning in dynamic environments that cannot be fully known or characterized in advance. Adaptation allows agents, whether they are robots, modules in an embedded system or software components, to change their behavior in response to environmental changes and actions of other agents in order to improve overall system performance. Additionally, adaptation allows swarms, artificial systems composed of large numbers of agents, to remain robust in face of failure even by a sizeable fraction of agents. If each agent had instantaneous global knowledge of the environment and the state of other agents, it could dynamically adapt to any changes in the environment or behavior of other agents. In most situations, however, such global knowledge is impractical or infeasible to obtain so one needs to devise different adaptation mechanisms based on partial, possibly noisy information about the state of the environment and the agents. Also, one would prefer a mechanism that would require little or no communication and/or negotiations between the agents.

Analysis is an important part of designing adaptive, self–organizing systems since it allows to understand global system properties given the behavior of individual entities and the rules of interactions between them. There are generally two options for the analysis of swarm behavior: experiment and simulation. Experiments with real agents, e.g., robots, allow the researcher to observe swarms under real conditions; however, experiments are very costly and time consuming. Simulations, such as sensor based simulations for robots, attempt to realistically model the environment, the robots' imperfect sensing of and interactions with it. Though simulations are much faster and less costly than experiments, they suffer from many of the same limitations, namely, they are still time consuming to implement, and systematically exploring the parameter space is often tedious. Mathematical analysis is an alternative to the time consuming and often costly experiments and simulations. Using mathematical analysis we can study dynamics of multi–robot systems, predict long term behavior of even very large systems, gain insight into system design, for instance what parameters determine group behavior and how individual robot characteristics affect the swarm. Additionally, mathematical analysis can be used to choose parameters that optimize the swarm's performance, prevent instabilities and so on. Note, however, that the mathematical analysis usually applies to strongly simplified systems, and it should be validated by comparing its results with the results of more realistic simulations (such as sensor based) and/or actual experiments with robots.

In this paper we present and analyze a simple stochastic model for adaptive task allocation in a team of robots, where robots have to forage for two distinct types of pucks, *Red* and *Green* scattered around the arena [5]. Each robot is able to collect pucks of a specific type, say *Red*: when a robot's foraging state is set to *Red*, it is searching for and collecting *Red* pucks. The goal of adaptive task allocation mechanism is to achieve a distribution of robots between two states that, over time, correctly reflect the pucks' prevalence. The robots have no global information about the the number of pucks of either color, or the states other robots. Instead, the robots make repeated local observations of the environment, store them in the memory, and use them to decide between two states. We analyze our model thoroughly using stochastic Master equation that describes the evolution of macroscopic dynamics, and compare it to the the results of discrete time simulations. We demonstrate that our analytical approach fully reproduces the results of the numerical simulations, suggesting that it might be used as an efficient tool for analyzing the global behavior in various behavior–based large–scale multi–robot systems.

2 Related Work

Mathematical analysis of the behavior of robots is a relatively new field with approaches and methodologies borrowed from other fields, such as mathematics, physics and biology. In recent years, a number of studies appeared that attempted to mathematically model and analyze collective behavior of distributed robot systems. These include analysis of the effect of collaboration in foraging [16, 17] and stick-pulling [9, 12] experiments, the effect of interference in robot foraging [7], and robot aggregation task [1, 6]. So far this type of analysis has been limited to simple reactive or behavior-based robots in which perception and action are tightly coupled. Such robots take input from sensors or behaviors and send output to actuators or other behaviors.[1] They make no use of memory or historic state information.

The role of learning in improving the performance of a multi-robot system has been addressed by several researchers. The RoboCup robot soccer domain provided a fruitful framework for introducing learning in the context of multi-agent and multi-robot systems. Several authors examined the use of reinforcement learning to learn basic soccer skills, coordination techniques [14] and game strategies [15]. Matarić [13] reviews research on learning in behavior-based robot systems, including learning behavior policies, models of the environment and behavior history. Goldberg and Matarić [2] present a framework for learning models of interaction dynamics in multi-robot systems. These models are learned on-line and used by robots to detect anomalies in system performance as well as to recover from these anomalies. Their work shares common foundation with ours: Markov processes as a model of interactions between robots . However, adaptation occurs as a result of the changing representation — the model of the interactions created and updated by robots — not as a result of changes in robot behaviors. Li *et al.* [10] introduced learning into collaborative stick pulling robots and showed in simulation that learning does improve system performance by allowing robots to specialize. No analysis of the collective behavior or performance of the system have been attempted in any of these studies.

Huberman and Hogg [3] studied collective behavior of a system of adaptive agents using game dynamics as a mechanism for adaptation. In game dynamical systems, winning strategies are rewarded, and agents use the best performing strategies to decide their next move. They constructed a mathematical model of the dynamics of such systems and studied them under variety of conditions, including imperfect knowledge and delayed information. Although the mechanism for adaptation is different, their approach, which they termed "computational ecology" is similar in spirit to ours, as it is based on the foundations of stochastic processes and models of average behavior.

3 Dynamic Task Allocation in Robots

Chris Jones and Maja Matarić presented an embodied simulation study of adaptive task allocation in a distributed robot system [5]. In their study, two distinct types of pucks, *Red* and *Green*, were scattered around the arena. Each robot could be tasked to collect pucks of a specific type, say *Red*. When a robot's foraging state is set to *Red*, it is searching for and collecting *Red* pucks. The robot can also recognize the foraging state of robots it sees. The robots have no *a priori* information about the shape of the arena, the number of pucks left in it or the number of foraging robots. The goal of adaptive task allocation is to design a robot controller that will allow robots to dynamically adjust their

[1] Robots that use timers to trigger actions can also be studied using this approach.

foraging type, so that the number of robots searching for *Red* and *Green* pucks will, over time, correctly reflect the pucks' prevalence.

The memory-based mechanism for adaptive behavior suggested by Jones and Matarić is consistent with the biologically-inspired control paradigm that has become popular in the field of distributed robotics. In such systems, the goal is to design local interactions among robots or between robots and the environment that will lead to desired collective behavior arises. The mechanism works as follows: As it wanders around the arena, robot counts the number of pucks of each type in the environment as well as the number of robots in each foraging state visible to it and adds these counts to memory. Since memory has a finite size new observations replace the oldest ones. Periodically, the robot uses values in memory to estimate the density of pucks and robots of each type, and changes its foraging state according to a certain transition function. The general idea is that a robot should switch its state to *Red* if there are fewer than necessary robots in the *Red* state and *vice versa* for *Green*.

In this paper we propose and study a slightly simplified model for task allocation, where the robots determine whether to make a transition to a new state based on the number of pucks of either types they encountered. Specifically, let m_r and m_g be the number of red and green pucks respectively that a robot has encountered in some time interval, so that the estimated fraction of red pucks is $\mu_r = m_r/(m_r + m_g)$. Then, the robot will choose the red and green states with probability μ_r and $1 - \mu_r$ respectively. Clearly, if the robots have global knowledge about the number of red and green pucks then this simple algorithm will achieve the desired distribution of the robots between the states. Hence, it is interesting to see how the incomplete knowledge about the environment affects this distribution, and in the case of dynamic environment (e.g., changing ratio of red and green pucks) what is its effect on the adaptive properties of the system.

4 Modelling Robots Observations

As we explained above, the transition rates between the states depend on robots' observations, or history. In our model, this history comprises of the number of red and green pucks a robot has encountered during a time interval τ. Let us assume that the process of encountering a puck is a Poisson process with rate $\lambda = \alpha M_0$ where α is a constant characterizing the physical parameters of the robot such as its speed, view angles, etc., and M_0 is the number of pucks in the arena. This simplification is based on the idea that robot's interactions with other robots and the environment is independent of the robot's actual trajectory, but are governed by probabilities determined by simple geometric considerations. This simplification has been shown to produce remarkably good agreements with experiments [11, 4].

Let M_r and M_g be the number of red and green pucks respectively, that generally can be time dependent, $M_r(t) + M_g(t) = M_0$. The probability that in the time interval $[t - \tau, t]$ the robot has encountered exactly m_r and m_g pucks is the product of two Poisson distributions:

$$P(m_r, m_g) = \frac{\lambda_r^{m_r}\lambda_g^{m^g}}{m_r!m_g!}e^{-\lambda_r-\lambda_g} \tag{1}$$

where $\lambda_i = \alpha\int_{t-\tau}^{t} dt' M_i(t')$, $i = r, g$, are the means the of respective distributions. In the case when the puck distribution does not change in time one has $\lambda_i = \alpha M_i \tau$, $i = r, g$.

5 Individual Dynamics

During a sufficiently short time interval, each robot can be considered to belong to a *Green* or *Red* foraging state. This is a very high level, coarse-grained description. In reality, each state is composed of several robot actions and behaviors, such as wandering the arena, detecting pucks, avoiding obstacles, *etc.* However, since we want the model to capture how the fraction of robots in each foraging state evolves in time, it is a sufficient level of abstraction to consider only these states. If we find that additional levels of detail are required to explain robot behaviors, we can elaborate the model by breaking each of the high level states into its underlying components.

Let us consider a single robot that forages for *red* and *green* pucks in a closed area and switches its state to *red* and *green* according to its observations. As a designer, we would like to define transition rules so that the fraction of time the robot spends in the *red* (*green*) foraging state be equal to the fraction of *red* (green) pucks. Let $p_r(t)$ be the probability that the robot is in the *Red* state. The equation governing its evolution reads

$$\frac{dp_r}{dt} = \varepsilon(1 - p_r)f_{g\to r} - \varepsilon p_r f_{r\to g} \tag{2}$$

where ε is the rate at which the robot has to make a decision whether to switch it state, and $f_{g\to r}$ and $f_{r\to g}$ are the corresponding transitions probabilities between the states. As we explained above, these probabilities depend on the robot's history, e.g., the number of either types of pucks it has encountered during the time interval τ preceding the transition. Specifically, let m_r and m_g be the number of red and green pucks respectively that a robot has encountered in that time interval. Then we define transition rates as follows:

$$f_{g\to r} = \frac{m_r}{m_r + m_g} \equiv \gamma(m_r, m_g), f_{r\to g} = 1 - \gamma(m_r, m_g) \tag{3}$$

Eq.2 is a stochastic differential equation since the coefficients (transition rates) depend on random variables m_r and m_g. Moreover, since the robot's history changes gradually, then the values of the coefficients at different times are correlated, hence making the exact treatment very difficult. Here we propose to study the it within the *annealed* approximation. Namely, we neglect the time–correlation between robot's histories at different times, assuming instead that at any time the real history $\{m_r, m_g\}$ can be replaced by a random one drawn from the Poisson distribution Eq. 1. Then, we can average Eq.2 over the histories to obtain

$$\frac{dp_r}{dt} = \varepsilon\overline{\gamma}(1-p_r) - \varepsilon(1-\overline{\gamma})p_r \tag{4}$$

where $\overline{\gamma}$ is the history–averaged transition rate

$$\overline{\gamma} = \sum_{m_r=0}^{\infty}\sum_{m_g=0}^{\infty} P(m_r, m_g)\frac{m_r}{m_r+m_g} \tag{5}$$

and $P(m_r, m_g)$ is the Poisson distribution Eq. 1. Note that if the pucks distribution changes in time then $\overline{\gamma}$ is time–dependent, $\overline{\gamma} = \overline{\gamma}(t)$. The solution of Eq. 4 subject to the initial condition $p_r(t=0) = p_0$ is readily obtained:

$$p_r(t) = p_0 e^{-\varepsilon t} + \varepsilon\int_0^t dt'\overline{\gamma}(t-t')e^{-\varepsilon t'} \tag{6}$$

To calculate $\overline{\gamma}(t)$ we define an auxiliary function

$$F(x) = \sum_{m_r=0}^{\infty}\sum_{m_g=0}^{\infty} x^{m_r+m_g}\frac{\lambda_r^{m_r}\lambda_g^{m_g}}{m_r!m_g!}e^{-\lambda_r}e^{-\lambda_g}\frac{m_r}{m_r+m_g} \tag{7}$$

so that $\overline{\gamma} = F(x=1)$. Differentiating Eq. 7 with respect to x yields

$$\frac{dF}{dx} = \sum_{m_r=1}^{\infty}\sum_{m_g=0}^{\infty} x^{m_r+m_g-1}\frac{\lambda_r^{m_r}\lambda_g^{m_g}}{m_r!m_g!}e^{-\lambda_r}e^{-\lambda_g}m_r \tag{8}$$

Note that the summation over m_r starts from $m_r = 1$. Clearly, the sums over m_r and m_g are de–coupled thanks to the cancellation of the denominator (m_r+m_g):

$$\frac{dF}{dx} = \left(e^{-\lambda_r}\sum_{m_r=1}^{\infty} x^{m_r-1}\frac{\lambda_r^{m_r}}{m_r!}m_r\right)\left(e^{-\lambda_g}\sum_{m_g=0}^{\infty}\frac{(x\lambda_g)^{m_g}}{m_g!}\right) \tag{9}$$

The resulting sums are evaluated easily (as the Taylor expansion of corresponding exponential functions), and the results is

$$\frac{dF}{dx} = \lambda_r e^{-\lambda_0(1-x)} \tag{10}$$

where $\lambda_0 = \lambda_r + \lambda_g$. After elementary integration of Eq. 10 (subject to the condition $F(0) = 1/2$), and using the expressions for λ_r, λ_0 we obtain

$$\overline{\gamma}(t) = \frac{1}{\tau}\int_{t-\tau}^{t} dt'\mu_r(t') + e^{-\alpha\tau M_0}\left(\frac{1}{2} - \frac{1}{\tau}\int_{t-\tau}^{t} dt'\mu_r(t')\right) \tag{11}$$

where $\mu_r(t) = M_r(t)/M_0$ is the fraction of red pucks. Eq. 6 and 11 fully determine the evolution of the dynamics of a single robot. To analyze its properties, let us first consider the case when the puck distribution does not change with time, $\mu_r(t) = \mu_0$. Then the we have

$$p_r(t) = \overline{\gamma} + (p_0 - \overline{\gamma})e^{-\varepsilon t} \tag{12}$$
$$\overline{\gamma} = \mu_0 + e^{-\alpha\tau M_0}(1/2 - \mu_0) \tag{13}$$

Hence, the probability distribution approaches its steady state value $p_r^s = \overline{\gamma}$ exponentially. Note that for large enough $\alpha\tau M_0$ the second term in the expression for $\overline{\gamma}$ can be neglected so that the steady state attains the desired value $p_r^s \approx \mu_0$. For small values of $\alpha\tau M_0$ (i.e., small density of pucks or short history window), however, the desired steady state is not reached, and in the limit of very small $\alpha\tau M_0$ it attains the value 1/2 regardless of the actual puck distribution (we elaborate on this more in Section 7).

Now let us consider the case when there is a sudden "jump" in the puck distribution at a certain time t_0, $\mu_r(t) = \mu_0 + \Delta\mu\theta(t - t_0)$, where $\theta(t)$ is the step function (without loss of generality we set $t_0 = 0$). Clearly, after some transient time, the distribution will converge to its new equilibrium value $\mu_0 + \Delta\mu$ (we assume that $\alpha\tau M_0$ is sufficiently large so we can neglect the exponential correction to the steady state value). After some simple algebra, we obtain from Eq. 6 and 11

$$\begin{aligned} p_r(t) &= \mu_0 + \frac{\Delta\mu}{\tau}t - \frac{\Delta\mu}{\varepsilon\tau}(1 - e^{-\varepsilon t}), && t \le \tau \\ p_r(t) &= \mu_0 + \Delta\mu - \frac{\Delta\mu}{\varepsilon\tau}(e^{-\varepsilon(t-\tau)} - e^{-\varepsilon t}), && t > \tau \end{aligned} \tag{14}$$

Eqs. 14 describe how the distribution converges to the new steady state value after the "jump". Clearly, the convergence properties of the solutions depend on τ and ε. It is easy to see that in the limiting case $\varepsilon\tau \gg 1$ the new steady state is nearly attained after time τ, $|p_r(\tau) - (\mu_0 + \Delta\mu)| \sim \Delta\mu/(\varepsilon\tau) \ll 1$, so the convergence time is $t_{conv} \sim \tau$. In the other limiting case $\varepsilon\tau \ll 1$, on the other hand, the situation is different. Indeed, a simple analysis of Eqs. 14 for $t > \tau$ yields $|p_r(t) - (\mu_0 + \Delta\mu)| \sim \Delta\mu e^{-\varepsilon t}$ so the convergence is exponential with characteristic time $t_{conv} \sim 1/\varepsilon$.

6 Collective Behavior

In this section we consider a collective behavior of a homogenous system consisting of N robots with identical controllers described in the previous section. Specifically, we are interested in the global system properties, namely, average number of robots in the given states and the fluctuations around this average. Note that the average number of robots in the red state is directly related to Eq. 4. Indeed, since the robots are in either state independent of each other, then $p_r(t)$ is simply fraction of robots in the *red* state, and consequently $Np_r(t)$ is the average number of robots in that state. Below we consider a more general problem of finding the probability distribution of having n robots in the red state.

Let $P_n(t)$ be the probability density that there are exactly n *Red* robots at time t. For a sufficiently short time interval Δt we can write [8]

$$P_n(t+\Delta t) = \sum_{n'} W_{n'n}(t;\Delta t)P_{n'}(t) - \sum_{n'} W_{nn'}(t;\Delta t)P_n(t) \tag{15}$$

where $W_{ij}(t;\Delta t)$ is the transition probability between the states i and j during the time interval $(t, t+\Delta t)$. In our multi robot systems, this transitions correspond to robots changing their state from red to green or vice versa. Since probability of having more than one robot to have a transition during a time interval Δt is $o(\Delta t)$, then, in the limit $\Delta t \to 0$ only transition between neighboring states are allowed in Eq. 15, $n \to n \pm 1$. Hence, we obtain

$$\frac{dP_n}{dt} = r_{n+1}P_{n+1}(t) + g_{n-1}P_{n-1}(t) - (r_n + g_n)P_n(t). \tag{16}$$

Here r_k is the probability density for having one of the k *Red* robots to changes its state to *Green*, and g_k is the probability density for having one of the $N-k$ *Green* robots to change their state to *Red*:

$$r_k = k(1-\overline{\gamma})\ ,\ g_k = (N-k)\overline{\gamma} \tag{17}$$

with $r_0 = g_{-1} = 0$, $r_{N+1} = g_N = 0$. Again, we have averaged the transition probabilities over the histories.

The steady state solution of Eq. 16 is given by [18]

$$P_n^s = \frac{g_{n-1}g_{n-2}...g_1g_0}{r_nr_{n-1}...r_2r_1}P_0^s \tag{18}$$

where P_0^s is determined by the normalization:

$$P_0^s = \left[1 + \sum_{n=1}^{N} \frac{g_{n-1}g_{n-2}...g_1g_0}{r_nr_{n-1}...r_2r_1}\right]^{-1} \tag{19}$$

Using the expression for $\overline{\gamma}$, we obtain after some algebra

$$P_n^s = \frac{N!}{(N-n)!n!}\overline{\gamma}^n(1-\overline{\gamma})^{N-n} \tag{20}$$

e.g., the steady state is a binomial distribution with parameter $\overline{\gamma}$. Note again that this is the direct consequence of the independence of robots' dynamics. Indeed, since the robots act independently, then in the steady state each of them has the same probability of being in either state. Moreover, using this argument it becomes clear that the time–dependent probability distribution $P_n(t)$ is given by Eq. 20 with $\overline{\gamma}$ replaced by $p_r(t)$, Eq. 6.

7 Simulations

To test the accuracy of our analytical formulation, we compared it to the results of discrete time numerical simulations with 100 robots. We model the arena by an 100×100 rectangular grid, M_r (M_g) cells are occupied by red (green) pucks. Robots move randomly from cell to cell[2], and once they are on a cell with either type of puck, they record it in their register. At each time step, each robot, with probability ε decides whether it should consider a transition or not, and then uses the transition rules described above to determine its new state, using the last τ entries in its registry. In Fig. 1 we plot the average fraction of red robots

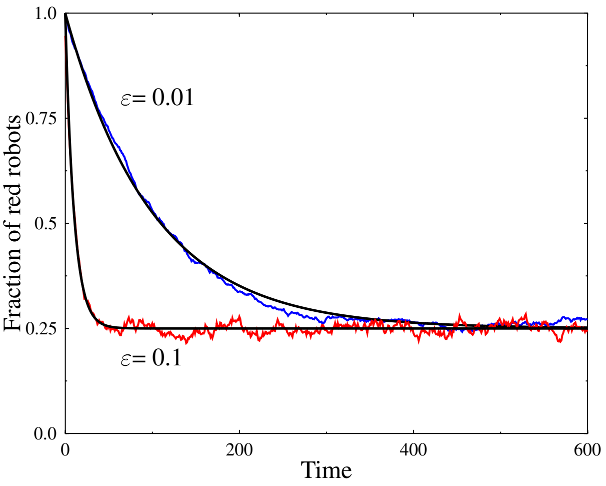


Fig. 1. Fraction of red robots vs time, $\tau = 50$

as a function of time for puck distribution $M_r = 500$, $M_g = 1500$, and for total number of robots $N = 100$, for different values of ε. We have averaged the plot over 100 trials. For comparison, we also plot $p_r(t)$ as given by Eq.12. One can see that the analytical curve fits perfectly with the results of the simulations. The fraction of robots in both cases converges to the same steady state value $p_0 = 0.25$, and the convergence time depends on ε as indicated by Eq.12.

The quality of task allocation depends not only on the average number of robots collecting, say, red pucks, but also the fluctuations around this average. Hence, we studied the steady state probability distribution. Clearly, the strength of the fluctuations are characterized by the width of this distribution. To obtain the steady state probability distribution in the simulations, we used the time series generated by a single run. To avoid the effects of transient dynamics, we carried out simulations until the steady state was reached, and then constructed

[2] Note that in our simulations we do not aim to reproduce realistic robot trajectories.

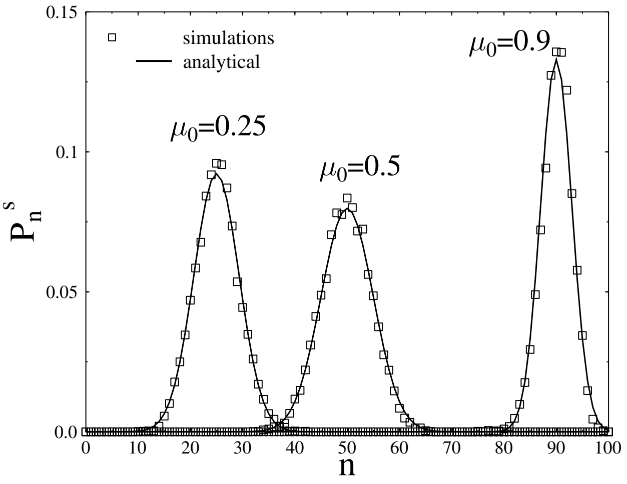


Fig. 2. Steady state distribution P_n^s for different fractions of red pucks

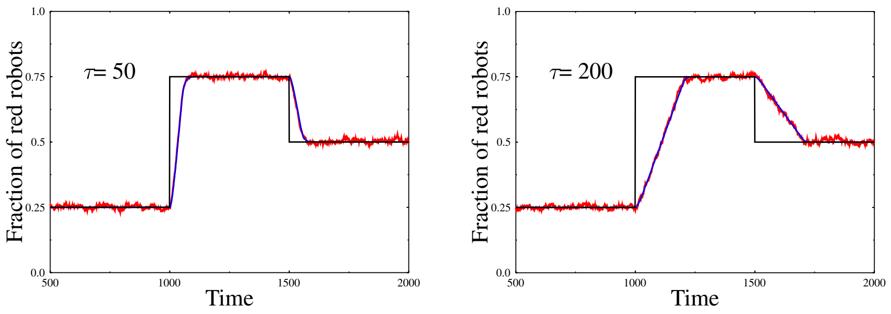


Fig. 3. Adaptation to changing puck distribution for different τ ($\varepsilon = 0.1$)

the histogram of $N_r(t)$–the number of red robots. The results are shown in Fig. 2 for different values of the fraction of red pucks. In each case, the distribution is peaked around its average value as one should expect. Again, one can see that there is an excellent agreement between the analytical curve (Eq. 20) and simulation results.

In Fig. 3 we plot the fraction of *Red* robots when the puck distribution undergoes step–like changes, both for simulations (averaged over 100 trials) and analytical results (Eqs. 14). One can see that the system adapts to the changes, and after some transient time the distribution of robots between the states reflects the puck distribution. Note that in this case also the analytical and simulation curves are virtually undistinguishable.

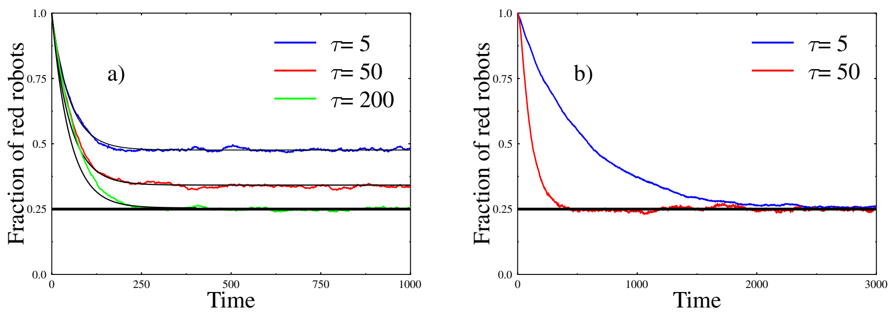


Fig. 4. a)Fraction of red robots vs time for different values of τ b)Fraction of red robots for modified transition rules. Both plots are the averages over 100 trials

Finally, let us consider the case when $\alpha\tau M_0$ is sufficiently small so that the correction to the value of $\overline{\gamma}$ can not be neglected. As we mentioned above, in this case steady state of Eq. 12 does not correspond to the puck distribution, $p_r^0 \neq \mu_0$, and in the limit $\alpha\tau M_0 \rightarrow 0$ the steady state converges to 1/2 not depending on μ_0. Note that this happens because for small enough $\alpha\tau M_0$ the robot's registry might not contain any readings at all. Hence, according to our rules,[3] each robot will choose either state with probability close to 1/2. This is illustrated in Fig. 4 (a) where we plot the number of red robots vs time for small overall density of pucks M_0/L^2 and different τ. Remarkably, the deviation from the desired steady state value is again well described by the analytical curve. Note also, that this undesired behavior can be avoided by modifying the transition rules as follows: if a robot's registry does not contain any reading for the last τ time steps, then the robot stays in its current state instead of choosing states with probability 1/2. This slight modification allows robots to achieve desired task allocation as shown in Fig. 4 (b).

8 Conclusion

In conclusion, we have presented a simple stochastic model of task allocation for multi–robot system, and studied it both analytically and in simulations. Dynamic task allocation model presented here is an adaptive form of foraging in a multi-robot system, where robots can switch dynamically between *Red* and *Green* foraging states. When a robot is in a *Red* foraging state, it is searching for and collecting *Red* pucks. The goal of dynamic task allocation is for the distribution of robots in *Red* and *Green* foraging states to dynamically adapt to the distribution of pucks, even when this distribution is not known in advance or changing in time. In order to accomplish this, robots make local observations of the foraging states of other robots and colors of pucks, estimate the densities

[3] Note that $\lim_{m_r \to 0} \lim_{m_g \to 0} \frac{m_r}{m_r + m_g} = 1/2$.

of each based on past observations, and switch foraging state according to some transition function. Transition function specifies the probability of switching state based on the observed densities of robots and pucks.

We have studied this model analytically using annealed approximation of stochastic Master equation, where the robot's actual histories are replaced by random one drawn from Poisson distribution. Although it is not clear a priori that such an approximation is valid, we obtained excellent agreement with the results of numerical simulations. Note also that the model presented here can be generalized to the situations when there are more than two states for more general multi–agent settings.

The work presented in this paper does not address the role noise in observations caused by faulty robot sensors plays in the behavior of the system. Real robots making observations have crude video systems and may not be able to distinguish two objects that are overlapping in their visual field, or even their types (colors). Nor can robots uniquely identify objects or be able to tell whether the object they are seeing has been observed before. Such limitations will often lead robots to overestimate or underestimate environmental states, and will require further elaboration of the analytical techniques described here. Capturing noisy observations and studying their effect on the collective behavior of an adaptive system is the focus of our ongoing research.

Acknowledgment

The research reported here was supported in part by the Defense Advanced Research Projects Agency (DARPA) under contract number F30602-00-2-0573. The authors would like to thank Chris Jones for introducing them to his task allocation system and useful discussions of the system behavior.

References

1. William Agassounon and Alcherio Martinoli. A macroscopic model of an aggregation experiment using embodied agents in groups of time-varying sizes. In *Proc. of the IEEE Conf. on System, man and Cybernetics SMC-02, October 2002, Hammamet, Tunisia*. IEEE Press, 2002.
2. Dani Goldberg and Maja J. Matarić. Coordinating mobile robot group behavior using a model of interaction dynamics. In *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 100–107, Seattle, WA, USA, 1999. ACM Press.
3. Bernardo A. Huberman and Tad Hogg. The behavior of computational ecologies. In B. A. Huberman, editor, *The Ecology of Computation*, pages 77–115, Amsterdam, 1988. Elsevier (North-Holland).
4. A. J. Ijspeert, A. Martinoli, A. Billard, and L. M. Gambardella. Collaboration through the exploitation of local interactions in autonomous collective robotics: The stick pulling experiment. *Autonomous Robots*, 11(2):149–171, 2001.
5. Chris V. Jones and Maja J Matarić. Adaptive task allocation in large-scale multi-robot systems. In *Proceedings of the 2003 (ICRA'03), Las Vegas, NV*. IEEE, 2003.

6. Sanza Kazadi, A. Abdul-Khaliq, and Ron Goodman. On the convergence of puck clustering systems. *Robotics and Autonomous Systems*, 38(2):93–117, 2002.
7. Kristina Lerman and Aram Galstyan. Mathematical model of foraging in a group of robots: Effect of interference. *Autonomous Robots*, 13(2):127–141, 2002.
8. Kristina Lerman and Aram Galstyan. Macroscopic Analysis of Adaptive Task Allocation in Robots. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS-2003), Las Vegas, NV*, Oct 2003.
9. Kristina Lerman, Aram Galstyan, Alcherio Martinoli, and Auke Ijspeert. A macroscopic analytical model of collaboration in distributed robotic systems. *Artificial Life Journal*, 7(4):375–393, 2001.
10. Ling Li, Alcherio Martinoli., and Yasser Abu-Mostafa. *Emergent Specialization in Swarm Systems*, volume 2412 of *Lecture Notes in Computer Science*, pages 261–266. Springer Verlag, New York, NY, 2002.
11. A. Martinoli, A. J. Ijspeert, and L. M. Gambardella. A probabilistic model for understanding and comparing collective aggregation mechanisms. In Dario Floreano, Jean-Daniel Nicoud, and Francesco Mondada, editors, *Proceedings of the 5th European Conference on Advances in Artificial Life (ECAL-99)*, volume 1674 of *LNAI*, pages 575–584, Berlin, September 13–17 1999. Springer.
12. Alcherio Martinoli and Kjerstin Easton. Modeling swarm robotic systems. In B. Siciliano and P. Dario, editors, *Proc. of the Eight Int. Symp. on Experimental Robotics ISER-02, Sant'Angelo d'Ischia, Italy*, Springer Tracts in Advanced Robotics 5, pages 297–306, New York, NY, July 2003. Springer Verlag.
13. M. J. Matarić. Learning in behavior-based multi-robot systems: Policies, models, and other agents. *Cognitive Systems Research*, 2(1):81–93, Apr 2001.
14. Martin Riedmiller and Arthur Merke. Karlsruhe brainstormers - a reinforcement learning approach to robotic soccer II. In *RoboCup-01: Robot Soccer World Cup V*, LNCS. Springer, 2001.
15. Peter Stone and Richard S. Sutton. Scaling reinforcement learning toward RoboCup soccer. In *Proc. 18th International Conf. on Machine Learning*, pages 537–544. Morgan Kaufmann, San Francisco, CA, 2001.
16. Ken Sugawara and Masaki Sano. Cooperative acceleration of task performance: Foraging behavior of interacting multi-robots system. *Physica*, D100:343–354, 1997.
17. Ken Sugawara, Masaki Sano, Ikuo Yoshihara, and K. Abe. Cooperative behavior of interacting robots. *Artificial Life and Robotics*, 2:62–67, 1998.
18. N. G. Van Kampen. *Stochastic Processes in Physics and Chemistry.* Elsevier Science, 1992.

Emergent Team Formation: Applying Division of Labour Principles to Robot Soccer

Tony White and James Helferty

School of Computer Science, Carleton University,
1125 Colonel By Drive, Ottawa, Ontario, K1S 5B6, Canada
arpwhite@scs.carleton.ca, stigmergy@hotmail.com

Abstract. Robotic soccer remains an area of active research owing to the difficulties of dynamic team formation and hard real time constraints regarding planning. Much of the existing research relies upon a central agency for coordination. Insect societies distribute work and allocate roles without a need for such a central agency and are robust with respect to changing environments and available agent resources. This paper explores the use of insect-inspired division of labour principles to robot soccer, highlighting the flexibility of the approach and ability to adapt to a wide range of soccer playing strategies.

1 Introduction

Insect societies are remarkable for their properties of self-organization. Work is assigned and roles allocated without a central coordinating agency. Insect societies naturally form teams; with individual agents being able to specialize in a particular role but being capable of role switching if environmental change demands it. For example, Polistes wasps consist of a single morphological caste, thereby providing no obvious physical reason for one wasp adopting a particular role or taking on a given task when compared to another. In social insects, the work of the colony is divided up into tasks. These tasks are then divided up amongst the individual insects in the colony. How they are divided up is a matter of considerable interest to agent researchers.

In nature, there are three main drivers in the way tasks can be divided up in a colony [1]:

1. *Temporal polyethism* – Individuals born in the colony around the same time period tend to perform similar tasks. It is not known if absolute aging is a factor, however, the social and external environment appear to influence behavioural development.
2. *Worker polyethism* – Individuals in different worker castes have different morphologies. The tendency is for workers within the same caste to tend towards similar types of tasks. Workers in different morphological castes tend to do different types of tasks.
3. *Individual variability* – Within a given caste, the individuals can develop heterogeneous behaviours, with some individuals tending more to working on one type of task than another. This is also referred to as a behavioural caste.

There is a certain level of plasticity in the examples of division of labour that occur in nature. Insect colonies have to be able to deal with factors such as perturbations in food availability, weather conditions, war, disease, etc. In nature, if a particular caste is wiped out, or its numbers reduced, the other castes will adapt their behaviour to take on the tasks previously performed by the other members. This is considered to be part of the division of labour problem.

Division of labour is a problem that presents itself naturally in the game of soccer. How many midfield and attacking players should I have on the team? Should my defenders ever adopt an attacking stance? Roles are usually allocated prior to the game and strategy is determined based upon an analysis of the opposition. Role reassignment occurs during breaks in the game, such as half time and stoppages in play. Role reassignment occurs through the substitution of players or movement of one player from one position to another. As such it is planned and does not occur in real time.

This observation motivates the contributions of this paper. Our desire is to make team formation a fluid and dynamic process. The team should have a set of possible roles for its players but should adopt as many attackers, midfield and defensive players as need be to best meet the strategic and tactics of the opposition. Further, should a team member get injured or tire, the team should compensate accordingly –all this without coordination by a central agency. It is our hypothesis that it should be possible to start with player roles unassigned and have them emerge during the game. Naturally, this is an ambitious goal and this paper represents first steps towards this goal.

In order to make significant progress, we have chosen to conduct our research in the simulation domain. We have chosen to use the TeamBots environment, which supports several different teams (with fixed roles), is written in Java and for which the source code was readily available.

The remainder of this paper consists of 4 sections. The next section briefly reviews the TeamBots environment and the teams provided. Following this, the division of labour and task allocation algorithms on which the research is based are described. Observations on the use of these algorithms in the TeamBots simulator follow. The paper ends with conclusions and proposals for future work.

2 TeamBots

TeamBots was designed and developed by Tucker Balch at Carnegie Mellon University, as a development and testing environment for studying collaborative robot behaviour. Originally titled JavaBots, due to its being written entirely in Java, the simulator is designed to provide a rapid prototyping environment for multi-robot logic development [2].

The language Java was chosen for three main reasons; portability, productivity, and modularity. Java interpreters are available on many operating systems, including those employed on the testing robots themselves, making it highly portable. Likewise, the researchers observed that they produced working code much more quickly in Java than they did C or C++, increasing their productivity. Lastly, the highly OO-nature of Java allows them the modularity they desired to be able to reuse the different logic and sensor modules they had developed in each of their own projects [2].

TeamBots is not the only soccer simulation environment available for use. The official RoboCup Soccer Simulator (RCSS) is, in fact, the more popular and well known, owing to its use in the yearly RoboCup tournaments [3]. However, RCSS teams tend to be written in C/C++ and targeted at Linux [4]. Since TeamBots was written entirely in Java, and the primary system for development was a Windows machine, TeamBots proved itself to be the more appealing choice.

TBSim is the title given to the TeamBots simulator. TBSim was designed such that it would simulate the operations of a variable number of robots on a field of arbitrary size. Obstacles and spherical objects are also simulated in 2D. A wall, for instance, is simulated as a line through which no robot may pass; a ball as an object capable of being kicked and pushed around, slowed by a constant decrease in velocity as it travels across the simulated ground.

The configuration of these objects is referred to as a Domain, which is read from a text file at the beginning of a run. Specified in this text file is a list of every object and obstacle in the Domain, along with their start positions and any other operating parameters necessary for initialization. Once a Domain is loaded into TBSim, a visual representation of the field appears, and simulation begins.

Due to the presence of Java interpreters as operating environments on actual robots, porting the logic of a TeamBots robot to hardware is, in theory, no more complex than the simple act of copying the file onto the robot. The interfaces provided for the robot logic in the simulator, TBSim, are identical to those offered in hardware, TBHard. There are no modifications that need to be made to the team's logic code whatsoever.

The interface provided by TeamBots from logic to hardware is completely abstract, and, as such, standardized across all supported robot kits; adding basic support for simulation of a new robot is as simple as adding logic to four function calls; getHeading, getPosition, setSpeed, and setSteer. Additional functionality beyond the basics is provided in the implementation of abstract Sensor and Actuator interfaces [2].

TeamBots supports the Probotics Cye as part of Carnegie Mellon's Minnow project [5].

3 Task Allocation and Specialization Algorithms

The algorithm Bonabeau [1] suggests a model of task specialization based upon a model insect division of labour; it is designed to model behavioural castes described in the introductory section. From an initially homogenous set of individuals, the result of the algorithm is to end up with a heterogeneous set of individuals, each member of which is specialized to a specific task.

In order to model this problem, each individual has a certain threshold for working on a task, as well as a stimulus for doing that task. The threshold lowers when they engage in that task (or learn it) and rises when they're not doing that task (forgetting it). Depending on the threshold value, the individual can have a greater or lessened probability of responding to the exact same level of stimulus.

The idea behind the algorithm is that individuals with more experience, and which are thus better equipped to handle a specific task, are more inclined to partake of that task than individuals who have less experience with that task.

The probability of an individual *i* undertaking a task *j* is expressed as:

$$T_{\theta_{i,j}}(s_j) = \frac{s_{i,j}^2}{s_{i,j}^2 + \alpha\theta_{i,j}^2 + \beta d_{i,j}^2}$$

Where $\theta_{i,j}$ is the self-reinforcing threshold for individual *i*, task *j*, and $d_{i,j}$ is the distance from individual *i* to where task *j* is performed. α and β are tuning coefficients, which we set to 1. (We are already normalizing, and thus weighting, $\theta_{i,j}$ and $d_{i,j}$.) Whenever individual *i* is performing task *j*, the self-reinforcing equation is:

$$\theta_{i,j} \leftarrow \theta_{i,j} - \xi\Delta t$$

Whenever individual *i* is not performing task *j*, the self-reinforcing equation is:

$$\theta_{i,j} \leftarrow \theta_{i,j} + \varphi\Delta t$$

The value of $\theta_{i,j}$ is restricted to between 0 and a maximum value. (We use 1)

For our implementation, we use values of $\xi = 0.00003$ and $\varphi = 0.00002$.

As the individual performs one task more than others, this causes the threshold for that task to drop, while the thresholds for other tasks increase. Since the probability function is based on the threshold, a lower threshold means a greater tendency to perform that task, reinforcing the selection of that task, thereby reinforcing the behaviour.

The use of a distance in the equation allows for us to give a higher probability to those individuals that are closer to the task performance location.

3.1 Implementation

The implementation consists of three main classes. The Specialization class provides an interface to the implementation, while the SpecializationConfig class provides instant access to configuration information. Internal to the implementation is the Task class, which represents a task within the context of the Specialization algorithm.

TeamBots allows one to specify a minimum frequency with which a robot will be given new information, and allowed to make decision changes in. By default, the simulator guarantees one of these cycles every 1/10th of a simulated second. TeamBots refers to these occurrences as “TakeStep” intervals.

Each time the robot is instructed to take a step, we must update the Specialization algorithm as to what is going on. For the algorithm to work, we must keep it appraised of the current level of all stimuli.

In general, we calculate these values as follows:

$$s_j = \sum_k [w_k \cdot s_k]$$

We restrict the values of the constant weight coefficients, w_k, such that

$$\sum_k w_k = 1$$

3.2 Basis Team for Specialization Algorithm

The team that the specialization algorithm was integrated into was called FemmeBotsHeteroG, and was originally written by M. Bernardine Dias of Carnegie Mellon University. This team was selected for its simplicity; the team has one "center," two "attackers," a "defender" and a "goalkeeper." While our algorithm retained the behaviours (roles) associated with the team, no one player was allocated to a given role.

The "center" hangs around the centre of the field, waiting until the ball comes within its sphere of influence. Then it heads straight for the ball. Once it is the closest one to the ball, the centre tries to get it as close as it can to the opponent's net, and potentially score. If it is no longer the closest one on its team to the ball, it returns to its wait position in the centre of the field.

The "attacker" waits around centre while the ball is in its own end, and attempts to score while the ball is in its opponent's end. There are two attackers, the north and the south. The only difference between the two is where on the field they wait, the north attacker stays just north of centre, while the south stays just south of centre. When attempting to score, the two work together to get the ball into the opponent's net. If they are unable to get it in by kicking, they work together, and use their brute strength to push any defender out of the way as they take the ball to the goal.

The "defender" waits near the goalkeeper for an opponent to enter the two-thirds of the field closest to its team's net. The defender then rushes to get into position between the ball and its net. If the defender is the closest one to the ball, and it has a clear shot, it kicks the ball towards the opponent's goal. If the ball travels into one of the close corners, it is the defender's job to brace against the goalkeeper so the ball can't be pushed into the net.

The "goalkeeper" waits at the net for the ball, tracking its movements back and forth (north and south) across the field, as far as the net goes. As the ball draws closer to the net, the goalkeeper continues to track the ball, attempting to punt it away from the net as it comes within range. Should the goalkeeper find itself out of the net, perhaps as a result of being pushed by an opposing player, it attempts to return to position as quickly as possible.

Since a goalkeeper is deemed necessary regardless of perceived demand, much like an insect queen to a nest, the goalkeeper is considered independently of the specialization algorithm.

3.3 Team Formation by Stimulus Calculations

Many previous approaches to team formation have involved a central agency for coordination or high level inter-agent dialogues [6, 7]. The stimulus equations given below require neither inter-agent communication nor a central coordinating agency. The equations are not strictly correct; we must ensure that the values of the stimuli stay within bounds at all times (0 to 1). Hence, before being subtracted from 1 and multiplied by their scaling factor s_k , each equation is corrected to maintain bounds; if it is less than 0, s_k is set to 0. If the equation's result is greater than 1, s_k is set to 1.

The stimulus values used in the implementation are as follows:

$$s_{center} = 0.5 \cdot \left[1 - \frac{|v\_ball\_to\_center|}{F\_DIAG \cdot 0.5} \right] + 0.5 \cdot \left[\frac{\sum_{l}^{teammates} [F\_DIAG \cdot 0.5 - v\_center_l]}{F\_DIAG \cdot 0.5} \right]$$

$$d_{center} = \frac{|v\_to\_center|}{F\_DIAG \cdot 0.5}$$

v_ball_to_center is a vector from the ball to the centre of the field. *F_DIAG* is a constant specifying the diagonal distance from one corner of the field to the other. v_center_t is a vector from team mate t to the centre of the field.

$$s_{attack} = 2 - \frac{|v\_ball\_to\_goal|}{F\_DIAG \cdot 0.5}$$

$$d_{attack} = \frac{|v\_ball\_to\_me|}{F\_DIAG}$$

v_ball_to_goal is a vector from the ball to the opponent's goal. *v_ball_to_me* is a vector from the ball to the robot. This stimulus equation turns 1 whenever the ball crosses the line into the opponent's territory, and gradually fades linearly to 0 as the ball travels closer to the home goal.

Since the north and south attackers are essentially the same, we treat them as one role. When assigning a robot to this role, we randomly switch between which to assign them to. This has the side effect of causing attackers to never truly settle into a position at the centre of the field, instead preferring to jump back and forth from the north rest position to the south.

$$s_{defender} = 0.5 \cdot \left[1 - \frac{|v\_ball\_to\_our\_goal|}{F\_DIAG} \right] + 0.5 \cdot \left[\frac{\sum_{l}^{opponents} [F\_DIAG - v\_net_l]}{\sum_{l}^{teammates} [F\_DIAG - v\_net_l]} \right]$$

$$d_{defender} = \frac{|v\_def\_zone|}{F\_DIAG}$$

v_ball_to_our_goal is a vector from the ball to the team's home goal. *v_def_zone* is a vector from the robot to the position where the defenders play. (In this case, the net.) v_net_l is a vector from robot l to the team's home goal.

The summations of the opponents and teammates are subtracted from each other to obtain a value that corresponds to the balance of defenders to the number of our opposition's attackers in the defensive zone. If there are more opponents in our end than there are teammates, this should spur the creation of more defenders to compensate.

3.4 Task Decisions

For each step, the stimulus is updated, and the specialization routine is checked to determine which role the player should be engaging in.

First, the routine checks if it is already engaged in a task. If it is, the routine checks to see if it has exceeded the minimum interval, $1/p$, before dropping a task. If it has, the task is dropped. We use a value of p=0.005, thus making the minimum interval 2000ms.

Next, the routine checks again to see if it is engaged in a task. If it isn't, then the routine goes through the process of selecting a new task. The process of determining a new task is equivalent to taking the probabilities of all of the possible tasks, and putting them on a number line. Then a random number r is rolled, and wherever that number falls on the number line is what the new task will be.

This is implemented as follows:

$$t = r \cdot \left[\sum_j \left[T_{\theta ij} \right] + T_{\theta_{idle}} \right]$$

where t is the target value we'll be looking for, and $T_{\theta_{idle}}$ is the idle probability constant. Our implementation uses a value of 0. We then loop through the tasks, adding up their probabilities, until we reach the task where the value of the thus-summed probabilities exceeds t. This is the task we select.

If we exceed the list of probabilities, then the target task is obviously the idle task. In this eventuality, we select no role for the individual i.

We repeat this process for as long as the robot is functioning.

4 Observations

4.1 The Opponents

All robot teams were tested using the simulator from the TeamBots 2.0e package. The description file used by the simulator was the exact same as the one supplied, excepting the portion pertaining to the class files of the robots to be used. SpFemmeBotsHeteroG (our specialization implementation of the FemmeBotsHeteroG team) was supplied in place of the name of the default team for the west side. The following robot teams were used as opponents on the east side: FemmeBotsHeteroG, DaveHeteroG, AIKHomoG, BrianTeam, CDTeamHetero, SchemaDemo, MattiHetero.

4.2 Interesting Behaviours

While testing against the various control teams, various trends in behaviour were observed. What follows is a list of these observations, along with examinations of their potential causes.

4.2.1 Centers

One of the most effective strategies of the specialization team was made possible through the exploitation of a deadlock-preventing feature built into the simulator. Whenever the ball's position is frozen for several seconds, a timer will run down and then the ball will be dropped again at the centre of the field. To exploit this behaviour of the simulator, the author of the original team coded the center so it would hover around the middle of the field, waiting for these ball drops. Attackers were also coded to hover in the same area when not in use, so as to be prepared to run in with the ball after a drop.

Freezing a ball is a relatively simple task. Since all that is required for the ball to be considered frozen is for its position to remain constant for a short period of time, there are two ways in which this can be made to occur; the ball can either be kicked into a position where it takes too long for a player to get to it before it times out, or the ball can be trapped between a player and another object on the field.

An effective strategy for defensive robots then, in removing the ball from their zone, is to either kick the ball into the corner, where it is unrecoverable (either through time restrictions in a robot travelling to it while it's at rest, or in its being left flush against the wall), or to push back with equal force against an oncoming opponent intent on scoring. If done effectively, the latter will cause the ball to become stuck between the two robots, and shortly thereafter, dropped in the centre of the field. A center is then able to quickly gain possession of the ball, and charge towards the opposition's goal.

In the specialization team, it was important to maintain the effectiveness of this tactic. By including a center-generating weighting value in the specialization equations in the specialization team, we ensure there remains an impetus on the team to leave at least one robot in the center role. As a result, it is relatively rare that the centre of the field not have a robot nearby engaging, or prepared to engage, in center role behaviour.

However, there are still times we can end up with an absence of players near the center. Fig. 1 shows a game against DaveHeteroG where the centre of the field has been left unprotected. The center that had been occupying that area had rushed back to take on a defender role temporarily, leaving no robots available to take ball possession following the drop. While in Fig. 2, the defender is able to make it to the ball before the others, and keep it inside the opposition's zone, we have lost field position as the other team has the time to get into a better defensive position in the middle of their zone.

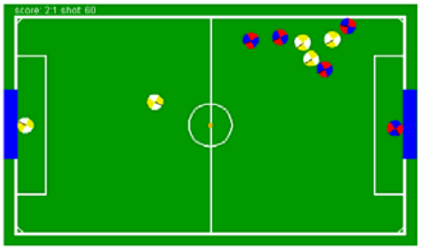


Fig. 1. (left) SpFemmeBotsHeteroG (light) versus DaveHeteroG (dark). No center; a defender comes forward to fill the void

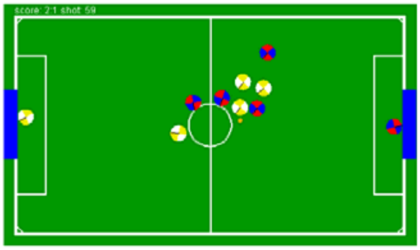
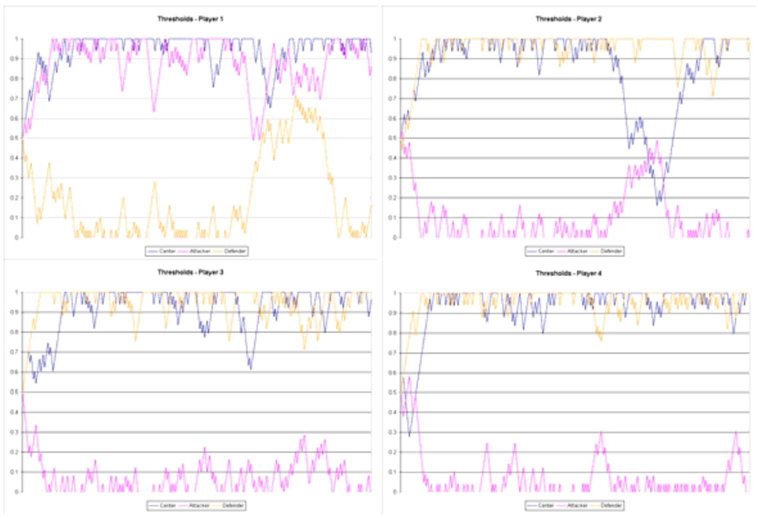


Fig. 2. (right) Having kicked the ball forward towards its teammates, the same robot turns to resume its duties as defender near the net

This occurs because even when the strength of the defender stimulus is low, if the threshold for a defender is low enough, there is still the random possibility that our robot will assume the role, regardless of demand. The defender stimulus drops whenever performing defender behaviour, which is sparked by insurgences by the other team into our zone. Hence, the probability of centers spontaneously becoming defenders is proportional to the probability of the opposing team having been deep in our zone sometime recently.

As we can see from the threshold graphs (Fig. 3-6), we ended up with no dedicated centers this game. However, from the various dips from all players, we can see that performance of this task was split relatively evenly amongst all involved. The exception is player two, which was close to becoming a dedicated center at one point. This coincides with a strong offensive by our team; even though player two was an attacker, it was the closest one to the centre of the field, and so it took on the role of center. The following return to previous levels was sparked by the center receiving possession of the ball, and bringing it into the opposition's zone. The attacker stimulus then overrode the center stimulus, and player 2 returned to being an attacker.



Figs. 3-6. (top left) Thresholds – Player 1. Primarily a defender; the downward spikes at the end for center indicate performance of the center behaviour, when the defender rushed forward to kick the ball to its teammates– (top right) Thresholds – Player 2. Primarily an attacker– (bottom left) Thresholds – Player 3. Primarily an attacker– (bottom right) Thresholds – Player 4. Primarily an attacker

Curiously, the degree of specialization towards being a center in this case is indicative of the all of the experiments run. Center specialization with this set of stimulus equations is exceptionally rare. Increasing the stimulus levels for the center relative to the others will lead to increased specialization as centers, and better odds of collection after a ball drop. However, the trade off is in effectiveness of defense, and the commitment of attackers, whose memberships tend to suffer as a result.

With more centers, the game becomes more back and forth, with an incursion into one zone, immediately followed by an incursion into the other. Defenders are not necessary with such tactics; for such a team, it becomes a matter of grabbing the ball as soon as it appears, rushing to get it close to the net, and then either making a lucky shot, or pushing their goalkeeper sideways until the ball falls into the net. So while a stronger center specialization does help the team win more games, it is by relying on the strengths of our team's goalkeeper, and the lack of exploitation of the same brute-force scoring techniques by the other team, rather than on the specialization algorithm itself.

The other flaw of having a more powerful center stimulus is due to the overlap in job description between a center and an attacker. A stronger center stimulus would have to be accompanied by a tighter description contributing to the stimulus; simply increasing the stimulus amount can lead to confusion on whether to perform a center or an attacker role. This confusion can manifest itself in a lack of commitment on the part of an attacker, who will turn away back towards centre while they are still relatively close to the ball, and deep and alone in the opposition's zone.

By having a weaker center stimulus, we are still able to generate a sufficient minimal number of centers because our attackers loiter in the same position. Thus, when the ball is dropped, they are close enough that the proximity will cause the center stimulus to override the attacker.

4.2.2 Task Changes

There are five players on each team. On the specialized team, four of these can become specialized in a particular task, of which we have three.

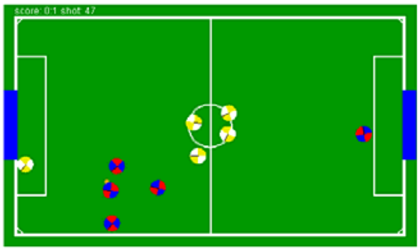


Fig. 7. SpFemmeBotsHeteroG versus SchemaDemo; all of the specialization individuals have become attackers

In this instance, we have had all four individuals on our team choose the attacker role at the same time, in response to the ball being in our opposition's zone. Shortly thereafter, the ball passes across the line, and moves into our zone, with all four of their attackers accompanying the ball.

Our attacker role does not specify any sort of defensive behaviour, leaving our goalkeeper alone to defend against the incursion. Even if several of our specializing players manage to become defenders, it may already be too late, as it would still take at least five seconds for the closest one to reach the goal.

Statistically, it is rare that all of our specialization players will choose to be an attacker when we need defenders. However, there are other more common scenarios with similar consequences.

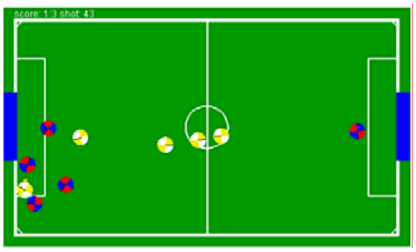


Fig. 8. (left) SpFemmeBotsHeteroG (light) versus SchemaDemo (dark); two defenders, two attackers

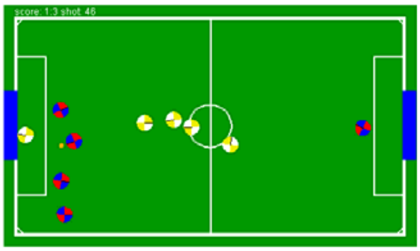


Fig. 9. (right) SchemaDemo preparing to score; two defenders, two attackers (two have swapped roles)

Fig. 8 illustrates a game between our team and SchemaDemo. The SchemaDemo team has made it into our zone, and the ball is between our goalkeeper and an opposing player and the net. Of our specialized players, there are currently two defenders, and two attackers. One defender is over halfway to the goal.

Fig. 9, however, shows what happens shortly thereafter. The closest defender has changed roles to an attacker, due to its low attacker threshold. At the same time, one of the attackers has changed roles to defender, due to the low ratio of defenders to opponents in our zone. While we now still have the same number of attackers and defenders, we have lost field position in a critical situation. The other team scores shortly hereafter.

This occurs because all of the players on our team have developed low attacker thresholds. If we spend a substantial amount of time in the other team's zone, then we will tend to have more dedicated attackers than defenders. This will happen because the stimulus equations are influenced by the amount of time spent performing the behaviour. Should the opposing team be able to dip in quickly, and score effectively each time they do, our team will be completely unprepared to defend itself against these attacks.

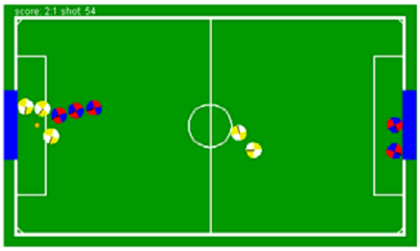
In this instance, even though all of the players on the opposing team are deep within our zone, and are preparing to score, our players still have a higher tendency to want to assume the attacker role than the defender role, due to their defender thresholds being substantially higher.

In an insect colony, there can be thousands of individuals available to do the work of the colony, while in a robot soccer team there are only four. Since the proportion of individuals to tasks to be performed is substantially lower, it stands to reason that indecisive behaviour such as this could be more apparent a detriment to the efficiency of this algorithm in a soccer team.

4.2.3 Goalkeeper Interference

An effective, and allowed, tactic in the TeamBots simulator is to prevent the goalkeeper from doing their job, by restricting their movement in front of the net. There are several teams included with the simulator that intentionally exploit this tactic. (e.g. Kezche, JunTeamHeteroG, etc.) Since the basis team was ill equipped to defend against these tactics, they were dropped from testing.

In practice, however, a goalkeeper interference tactic may end up being unintentionally leveraged by an aggressive opponent. Fig. 10 demonstrates a scene from a game between our specialization team and its basis team. In this instance, the goalkeeper is being prevented from doing its job because a defender is in its way, which is in turn being blocked by an opposing player. (This can also happen if we have more than one defender, and they are both fighting for the same position.)



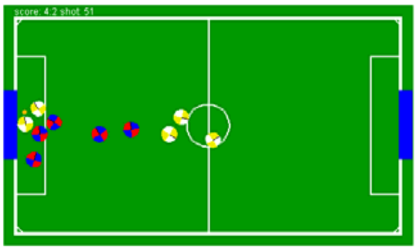


Fig. 10. (left); SpFemmeBotsHeteroG (light) versus FemmeBotsHeteroG (dark); the goalkeeper, blocked by defender, blocked by attacker from the other team, as the ball trickles into our net

Fig. 11. (right); SpFemmeBotsHeteroG (light) versus MattiHetero (dark); the goalkeeper is blocked by a defender which has just turned into an attacker

Unwise specialization decisions in our own team can be just as problematic. Fig. 11 shows a scenario where the opposing team is about to score because our goalkeeper is being blocked. In this instance, it is the fault of the player in front of the net, which has just changed to the attacker role. The attacker makes no attempt at avoiding the ball, or at giving way to the goalkeeper as it turns to pursue its new role at the centre of the field.

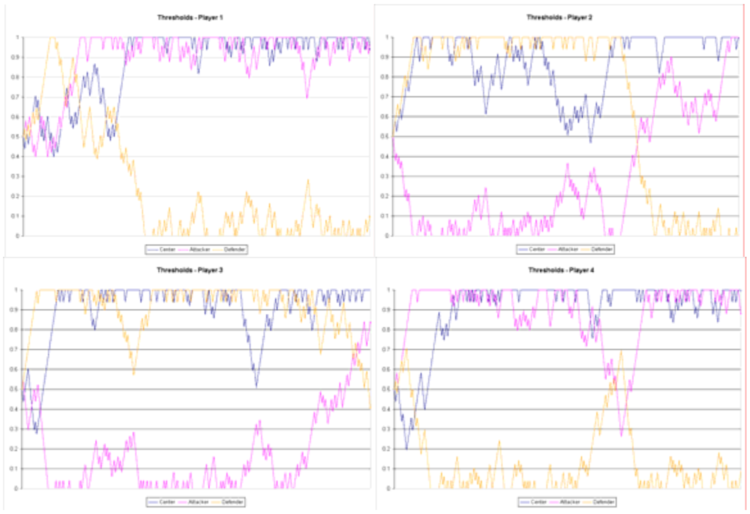
Also visible in Fig. 11 is an effective tactic used by the MattiHetero team (and also by AIKHomoG) in preventing goalkeeper interference tactics from being used against its team; the goalkeeper does not remain against the net, but rather stays out from the net. In this way, if an opposing player attempts to push them out of the way, they will still have the ability to pull back and around them, and into position.

4.2.4 Defensive Play Against Aggressive Opponents

If the team is performing particularly badly, and the other team is in our zone for a large part of the game, this can inspire more team members to become defenders. Fig. 12-15 show the resultant threshold values of play against AIKHomoG. This team is exceptionally aggressive, and spends most of its time in our zone. Our team attempts to balance against this configuration.

As we can see from the graphs, player four becomes a defender almost immediately, followed by player one and player two. As the game ends, we can see player three becoming a defender as well.

The key to this opponent's success is in the two players it leaves just outside the corners of our net, whose only purpose is to score. This team is purely an offensive team, which makes them a difficult competitor for the specialization team. On the other hand, their defence is relatively weak. The center-heavy configuration mentioned previously put in a better showing against this opponent for just this reason; when attacked en masse, AIKHomoG's lone goalkeeper puts up little resistance.



Figs. 12-15. (upper left) Thresholds – Player 1; SpFemmeBotsHeteroG versus AIKHomoG. – (upper right) Thresholds – Player 2 – (bottom left) Thresholds – Player 3 – (bottom right) Thresholds – Player 4

4.2.5 Delay of Game

The ball drop on ball freeze feature of the simulator can further be exploited for delay of game. Should the ball ever be re-dropped in the middle of three robots ringed around the centre, the ball is essentially frozen permanently, and whomever was winning up until that point will win the game.

While no teams were observed to actively exploit this flaw in the simulator, it is conceivable that a team could be designed -- or possibly evolved through a genetic algorithm -- to exploit this flaw in the simulator, freezing the ball in the centre of the field while they are in the lead and ensuring their victory.

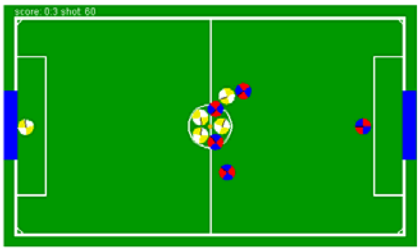


Fig. 16. SpFemmeBotsHeteroG (left) versus SchemaDemo (right); the ball is frozen for the rest of the games, as the two teams refuse to yield, and allow the other team room to knock the ball from the centre after a ball drop

5 Conclusions

The implementation presented demonstrates several fundamental strengths and weaknesses in applying the specialization division of labour algorithm to a simulated robot soccer team.

The time spent travelling between roles can be an impediment to the performance of the team. Since the roles in the basis team perform their behaviours in different general areas on the field, transit time from one area to another after a role decision becomes a factor.

This problem can be compounded if we accidentally choose a poor role, which does happen quite frequently. With this algorithm, position on the field does not matter to an individual to the same degree as what role the robot was engaged in most recently for the longest period of time. As such, players may miss good opportunities because they are more inclined to perform the roles they're more comfortable with.

One possible improvement to fix this problem would be a further refinement of the stimuli for the various roles on the field. For example, a dampening value might be subtracted from the defender stimulus when the player is currently in possession of the ball, and close to the opposition's goal. The challenge of such a refinement would be in ensuring a balanced set of dampening values for all roles, so that one role doesn't get dampened more harshly or leniently than the rest. We also believe that adding a velocity term into the stimulus equations will influence performance. Another potential improvement would be to learn the stimulus equations using Genetic Programming.

The basis team tended to be hardwired for its tasks with a specific number of robots (one) per role in mind. As such, it did not scale well to a varying number of individuals in the defender position, for instance, where extra defenders tended to get tangled up in each other.

In this implementation, our tuning coefficients α and β for our probability equations were simplified to 1. A further refinement of this implementation may be to experiment with these values, to see what effect modification of them might have on the ranges at which individuals participate in team behaviour.

Furthermore, a team balancer stimulus might be employed to give the team a more aggressive or defensive posture. (This global team reinforcement strategy has been suggested and employed before by Balch [8].) If the team is losing, then it may make sense for the defender's stimulus to be added to a reinforcement value, which would spur the creation of defenders even whenever there are no attackers in our half of the field.

Likewise, if the opposition is spending a large portion of time in our half of the field, then perhaps we aren't being aggressive enough, and we need more centers to hold the line when the ball reappears in the middle. In situations like this, an aggression stimulus could be calculated and added to the regular center stimulus, to react to a perceived need for more centers.

The key strength of the specialization algorithm is that it is capable of adapting to changing conditions on the field. If we need more defenders, this algorithm remembers that fact on an individual-by-individual basis. Even if the ball goes into our opposition's zone, an individual will remember, for some time after that, to stay back so as to be prepared. In this respect, the specialization team performed its task as expected.

References

1. Bonabeau, E., Dorigo, M., Theraulaz, G. (1999) Swarm Intelligence, Oxford University Press, pp. 110-143.
2. Balch, T., Ram, A. (1998) Integrating Robotic Technologies with JavaBots, Working Notes of the AAAI 1998 Spring Symposium, Georgia Institute of Technology.
3. Anon. (2003) The RoboCup Soccer Simulator, http://sserver.sourceforge.net/
4. Cisternino, A., Heintz, F. (2003) The RoboCup Simulator Team Repository, http://medialab.di.unipi.it/Project/Robocup/pub/
5. Stancliff, S. (2001) The Minnow Project, http://www-2.cs.cmu.edu/~coral/minnow/
6. Nair, R., Tambe, M., and Marsella, S. (2003) Role Allocation and Reallocation in Multi-agent Systems, In Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems.
7. Pynadath, D.V. and Tambe, M. (2002) The Communicative Multi-Agent Team Decision Problem: Analyzing Teamwork Theories and Models. Journal Of Artificial Intelligence Research, Volume 16, pp. 389-423.
8. Balch, T. (1997) Learning Roles: Behavioral Diversity in Robot Teams, GIT-CC-97-12, Georgia Institute of Technology.

Analyzing Stigmergic Learning for Self-organizing Mobile Ad-Hoc Networks (MANET's)

H. Van Dyke Parunak and Sven A. Brueckner

Altarum Institute,
3520 Green Court, Ann Arbor, MI 48105, USA
{sven.brueckner, van.parunak}@altarum.org

Abstract. In recent years, mobile ad-hoc networks (MANET's) have been deployed in various scenarios, but their scalability is severely restricted by the human operators' ability to configure and manage the network in the face of rapid change of the network structure and demand patterns. In this paper, we present a self-organizing approach to MANET management based on stigmergic agents and demonstrate how to analyze its performance under different deployment assumptions. Our results emphasize the importance of attention to notions from dynamical systems theory in designing and deploying multi-agent systems.

1 Introduction

The challenges of managing mobile ad-hoc networks (MANET's) [1] may overwhelm traditional network management approaches. Such networks are highly dynamic, severely constrained in their processing and communications resources, distributed and decentralized. Thus, centralized management approaches requiring accurate and detailed knowledge about the state of the overall system may fail, while decentralized and distributed strategies become competitive.

We have successfully applied fine-grained agent architecture modeled on algorithms used in biological systems [11] to a range of real-world problems, including manufacturing control [2], pattern recognition in sensor networks [4], collaboration and task assignment among multiple mobile platforms [13], path planning for unmanned vehicles [16], and information retrieval in massive data [17]. This paper explores the applicability of these mechanisms to another domain, mobile ad-hoc communication networks (MANET's). Like other domains in which swarming is effective, MANET's are distributed, decentralized, and dynamic. Self-organizing systems of agents with emergent system-level functions offer an approach that is robust, flexible, adaptive and scalable. By applying our techniques to a new domain, we gain experience with their capabilities and restrictions, and further exercise the development methodology that we are developing for such systems [12, 15].

Section 2 presents a concrete management problem in the MANET domain. Section 3 offers a solution based on fine-grained agents dynamically interacting in the network environment. Section 4 offers experimental evidence for the effectiveness of our solution. Section 5 concludes.

2 The MANET Server Management Problem

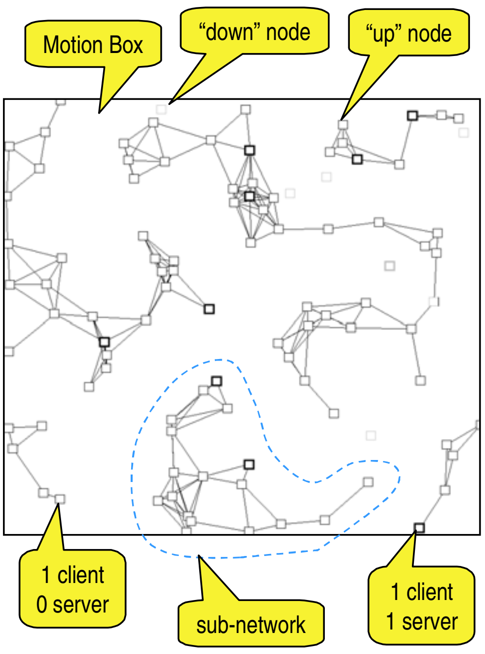


Fig. 1. Domain Overview

Figure 1 offers an overview of the MANET domain. Assume a network of (randomly) moving nodes that may communicate within a limited range, and may fail temporarily. A canonical example of an application for a MANET is a fleet of vehicles (say, trucks or dismounted troops in a military operation, or rovers exploring a remote planet) equipped with line-of-sight radios.

We focus our attention on configurations in which nodes may host distinct client and server processes. Every node carries a client and some nodes carry a server process. Examples of services that might be restricted to some vehicles include

- long-range communications links back to a remote commander;
- wide-range sensors that can provide an integrating context for more local sensors carried on most vehicles;
- target recognition databases and data fusion capabilities that can provide interpretive support for platforms with more local access.

A server provides a stateless and instantaneous service to a client upon request if there exists a communications path between the client and the server and if the server node is currently active. Servers in our model have no capacity constraints, and may serve as many clients at the same time as requests arrive.

Because the nodes are mobile, weight and space are constrained, limiting the power available for communications and processing. Some of the likely services (long-range communications or sensing) impose especially high power demands on the servers, making it desirable to operate them only when they are needed to support the demands from the rest of the fleet. Vehicle movement must satisfy two constraints: achieving mission objectives and maintaining communication connectivity. In the simple example we describe here, all vehicles share both objectives, but techniques that we have demonstrated elsewhere [13] permit vehicles to specialize for different tasks, so that some vehicles would dedicate themselves to serving as communication relays, reducing the constraints on the other vehicles imposed by the need to maintain connectivity.

The server management problem requires answering three questions: given the current topology of the network determined by node locations, communications ranges and node availability, decide

1. which server nodes should actually expend battery power to execute the server process;
2. to which server node a particular client should send its next service request; and
3. where to relocate server nodes to meet the current demand by the clients.

Thus, the network must be provided with mechanisms that self-diagnose the current network state (e.g., breaking of connections, availability of new connections, failure of nodes) and provide the information in a way that enables it to self-configure the ongoing processes appropriately. These functions could be satisfied if all servers executed constantly and if all clients had global knowledge of the overall system (Figure 2), but such a solution is impractical.

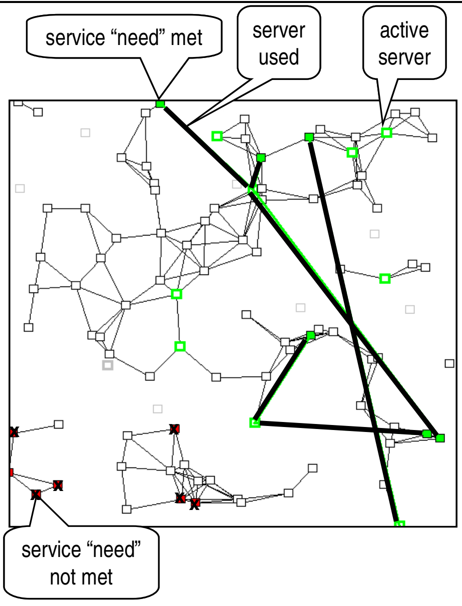


Fig. 2. Global Solution

3 Emergent MANET Management

A fine-grained, self-organizing agent system can solve the service location problem specified in Section 2. Our solution starts with the following initial conditions:

- Server processes shut down immediately if no requests arrive.
- A client does not know about the location of servers in the network, unless the client is co-located with a server on the same node.
- Server nodes move randomly (a zeroth order approximation to mission-motivated movement).

Thus, in terms of our design goals, we preserve maximum battery power, but most clients' service needs are not met since they don't know which server to address.

We now define a co-evolutionary learning process based on individual reinforcement. This learning process has three components.

1. The server population learns to maintain an appropriate number of active server processes,
2. and to adjust the position of these processes as they learn about the clients who are using them.
3. The client population learns to direct requests to active servers.

3.1 Server Activation Learning

Any server node is aware of the incoming requests from one or more clients. If the server process is running, then these requests are served, otherwise they fail, but the node will immediately start up the server process to be available for any new requests in the next cycle. While the server process is running, it tracks the number of incoming requests. If there are no requests, it will begin a countdown. It will either abort the countdown if new requests arrive (and are served), or shut down if it reaches the end of the countdown.

Initially, the duration of the countdown is zero. Thus, server processes are shut down as soon as no new requests come in. We define the following simple reinforcement learning process to adjust the duration of the next countdown:

(+) If a request from a client arrives and the server process is down, we increase the length of the countdown period for subsequent countdowns, since apparently the server should have been up and we lost performance (failed to serve a request) while the server was down.

(–) If no request arrives while the countdown proceeds and the server process reaches the end of the countdown, then we decrease the length of the countdown period for subsequent countdowns, since apparently the server could have been down already and we wasted resources (battery power) while the server was up.

Driven by the demand pattern as it is perceived at the particular server node, the server process learns to maintain the optimal availability. In effect, the server learns the mean time between requests and adjusts its countdown length accordingly to stay up long enough. With this learning mechanism in place, the client population will now assume the role of the teacher as it generates a demand signal that leads some servers to stay down (extremely short countdown) while others stay consistently up (extremely long countdowns).

3.2 Client Preference Learning

Initially, only clients that are co-located with a server on the same node have any information about possible server addresses. These clients will become the source of knowledge of the client population as they share this information with their neighbors.

Knowledge Representation—Clients manage their knowledge about and evaluation of specific servers in a dynamic set of scorecards, one for each server they know. A scorecard carries the address of the server, a score in favor (*pro*) and a score against (*con*) using this server. The current score of a server is computed as *pro - con*.

Decision Process—When a client needs to select a server, it normalizes the current scores of all scorecards so that they add up to one and selects a server with a probability equal to the normalized score (roulette wheel selection). Thus, servers with a low current score compared to others have a lower probability of being chosen by the client. If the client currently does not have any scorecards, then it can only contact a server if it co-located with one, otherwise its service need will not be met in this decision cycle.

Information Sharing—If a client selects a server on a node that is currently within reach, it sends a request to the server and shares the outcome of this interaction with its direct neighbors. If the request is met, the client increases its own *pro* score of that server by one and sends the same suggestion to its direct neighbors. If the request is not met, the *con* scores are increased in the same way. These suggestions to the neighbors may lead to the creation of new score cards at those neighbors if they had not known about this server before. Thus knowledge about relevant servers spreads through the network driven by the actual use of these servers. Furthermore, the success or failure of the interaction with a server reinforces the preferences of the client population and thus (with a random component to break symmetries) dynamically focuses the attention on a few active servers while encouraging de-activation for others (see "Server Activation Learning").

Truth Maintenance —The constant change of the network topology, driven by the node movements and their failures, requires that the client population continuously update its knowledge about reachable servers and their evaluation. While the score-sharing mechanism ensures that the performance of a reachable server is continuously re-evaluated, the clients still need a mechanism to forget references to servers that do not exist anymore or that are out of reach now. Otherwise, in long-term operation of the system, the clients would drown in obsolete addresses.

A client "evaporates" its scores (*pro* and *con* individually) by multiplying them with a globally fixed factor between zero and one in each decision cycle. Thus, both scores approach zero over time if the client or its neighbors do not use the server anymore. If both scores have fallen below a fixed threshold, then the scorecard is removed from the client's memory – the client forgets about this server.

A client also chooses to forget about a particular server, if the *con* score dominates the *pro* score by a globally fixed ratio (con / ($con + pro$) > threshold > 0.5). Thus, servers that are trained by the client population to be down are eventually removed from the collective memory and are left untouched. They only return into the memory of clients if all other servers have also been forgotten and their co-located client is forced to use them.

3.3 Server Node Location Learning

In a co-evolutionary process, the server and client populations learn which clients should focus on which servers. We can stabilize this preference pattern and reduce the need for re-learning by decreasing the likelihood that the connection between a client and its chosen server is disrupted. Since the risk for a disruption of the path between a client and a server generally increases with the distance between their nodes, moving the server node towards its current clients will decrease this risk.

We assume that any client and server processes have means to estimate their respective node's current spatial location and that the server node may actually control its movement within certain constraints if it chooses to.

As a client sends a request to a server, it includes its current location in the request message. The server node computes the vector between the client and the server location and adds up all vectors from all requests within a decision cycle. Vectors of requests that failed are negated before they are added to the sum. The resulting combined vector determines the direction of the next move of the server node. If the requests failed because the server process was down, then the node moves away from the "center of gravity" of the clients that contacted this server. Otherwise, the node will move toward these clients. The length of the step for the server node is fixed to a global constant, characterizing the physical ability of the node to move.

3.4 Stigmergic Coordination

The coordinated behavior of many simple agents (server, client, node) in the highly dynamic and disruptive MANET environment emerges from peer-to-peer interactions in a shared environment driven by simple rules and dynamic local knowledge. The individual components of the system are not explicitly aware of the overall system functions of self-diagnosis and self-reconfiguration.

The coordination mechanism detailed in this demonstration is an example of stigmergy, in which individual agent activity is influenced by the state of the agent and its local environment. As agent activity manipulates the environment, subsequent agent activity dynamics may change (Figure 3). If this flow of information between the agents through the environment establishes a feedback loop that decreases the entropy of the options of the individual agents, then coordinated behavior emerges in the population. We engineer the agent behavior and the indirect information flow, so that the emergent coordinated behavior meets the design goal.

Three populations of processes (agents) contribute to the emerging system functionality. Because each population operates in the shared network environment, the other populations influence its dynamics. For instance, the clients coordinate their server choice through the exchange of scores, but their ability to focus on only a few servers depends on the server population's ability to identify the emerging intention of the clients and to maintain the server processes on the correct nodes. Figure 4 identifies the main flow of information among the three populations driven by their respective dynamics and linked by the occurrence of successful or failed utilization events – requests from clients to servers.

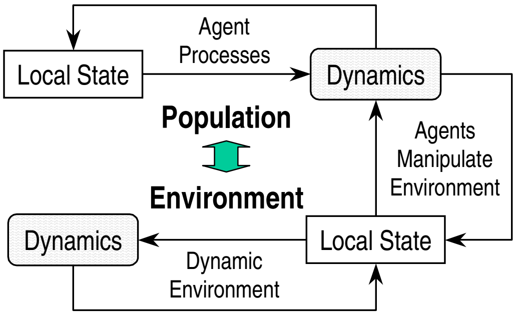


Fig. 3. Stigmergic Coordination (general schema)

A common feature of the server activation learning and client preference learning in our scheme is the combined reinforcement and decay of a critical decision parameter (the countdown on the server; pro and con scores on the server scorecards maintained by clients). Elsewhere [14] we describe this sort of process as "pheromone learning," because it combines two of the hallmarks of insect pheromones: periodic deposits, and constant background evaporation. Pheromone learning can be viewed as reversing the traditional approach to truth maintenance. Rather than maintaining any knowledge until it is proven wrong, we begin to remove knowledge as soon as it is no longer reinforced. This approach is successfully demonstrated in natural agent systems, such as ant colonies, where information stored in pheromones begins to evaporate as soon as it is laid down.

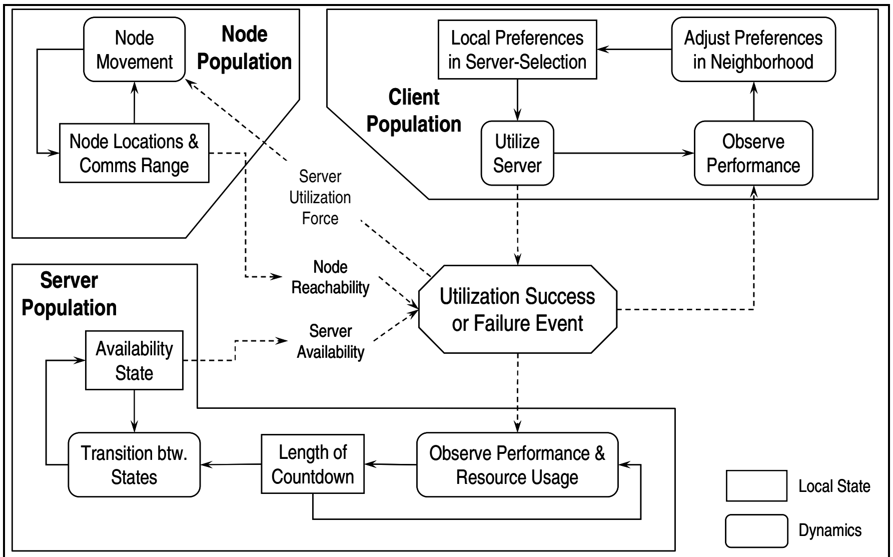


Fig. 4. Stigmergic Coordination in MANET's

4 Performance Analysis

As engineers, we need not only to conceive innovative architectures to address challenging real-world problems, but also to analyze these architectures to determine their performance as a function of deployment conditions. Such analysis requires three elements: a baseline against which to compare the performance of the innovation, a set of metrics to make this comparison, and experiments to apply the metrics to the new system.

4.1 Baseline

Baselines for performance evaluation can be of two kinds. Sometimes we have performance data for a conventional system and wish to show how our system compares with it (a relative evaluation). In other cases we have an upper bound on performance, a bound that may not be achievable in practice, but that shows how close to the theoretically best performance our solution (or any other) comes (an absolute evaluation).

In the case of MANET's, we can define a global solution that provides the highest possible request-success rate for the clients. We ignore the desire to preserve battery power and let all server nodes execute the server process at all times (maximum server availability). We use global knowledge (requiring very large bandwidth) to determine for a client that wants to send a request, which available server nodes are currently in range (path exists), and then we select the recipient of the request randomly from this set.

This solution formally avoids sending requests to servers that are out of reach, whose node is currently down, or whose server process is currently not executing. But its resource requirements are too large to meet the severe constraints of the application domain (ad-hoc mobile wireless network among battery-powered nodes). Also, from a more programmatic point of view, this solution does not demonstrate emergent cognition, since the complexity of the individual node (client) is as high as the system-level complexity. Nevertheless, this solution provides us with a performance and resource-usage baseline against which we measure our local approach in the demonstration.

4.2 Metrics

We focus our attention on two metrics of a system under a particular set of deployment constraints: resource gain and performance loss. Both are ratios comparing a key system-level feature with the baseline.

Resource gain describes the percentage of servers that our mechanism keeps on standby, that would be running and burning power in the baseline. The total number of servers is a constant in this scenario, and all of them are running in the baseline. So resource gain is directly proportional to the total number of servers on standby.

Performance loss measures the failure of service events in our mechanism compared with the baseline. Let

N = total number of service requests

N_b = total number of requests satisfied by the baseline;

N_t = total number of requests satisfied by the test system.

Since the baseline is the best possible in any given circumstance, $N_t \leq N_b \leq N$. Performance loss is defined as $(N_b - N_t)/(N - N_b)$. Unlike resource gain, performance loss is compared against a changing baseline, since N_b varies with system configuration, so we also track raw performance of our scheme.

4.3 Comparison with the Global Solution

With a baseline and metrics in hand, we can explore the performance of our system. The following discussion is meant to be exemplary, not exhaustive. We explore the variation in metrics as a function of three network characteristics: the degree of connectivity, the dynamics of individual servers, and the overall demand from the clients. Error bars in the plots are at ± 1 standard deviation, adjusted to avoid unphysical values (e.g., probabilities outside of [0,1]).

4.3.1 Configuration

Our experiments use a population of 100 nodes, of which 25 can serve as servers. They are initially distributed randomly in an arena sized 100 x 100, so the average area per node is 100, with radius ~5.6, and a mean internode separation on the order of 11. At each time step, several parameters determine the dynamics of the system.

- Range is a measure of the communications range of the nodes, in the same units that define the dimensions of the virtual world within which the nodes are distributed. The default setting is 15, which is greater than the mean internode separation of 11.
- DownProb (p_d) is the probability that a node will go out of service due to failure. The default setting is 0.02.

- UpProb (p_u) is the probability that a failed node will resume operation. The default setting is 0.90.
- UtilizationRate is the probability that a given node requests service. The default setting is 0.50.
- NodeMovementPolicy can be either directed (in which case servers and clients implement the algorithm outlined in Section 3.3) or random (in which case the direction of movement is chosen randomly, as a zeroth-order approximation to mission movement).
- ClientStepLength and ServerStepLength define the distance (in the same units as Range) that a node moves in adjusting its location under either movement policy. The defaults are 0.5 and 3.5, respectively.

4.3.2 Impact of Demand

Adaptive schemes such as ours require a steady stream of information about the environment, which in our case is provided by the success or failure of service requests. When service requests are at a very low level, the system cannot adapt effectively, reflected in the performance changes. Figure 5 shows the impact of changing utilization. All other parameters are fixed at their default values.

The mean value of raw performance increases with utilization, and performance loss decreases, but the error bars show that these changes are swamped by noise. It is important to note that the variance is much greater for low utilization (10%) than for the higher levels. At low utilization, the algorithm does not get sufficient information to make useful decisions, but at higher utilization levels, its behavior converges.

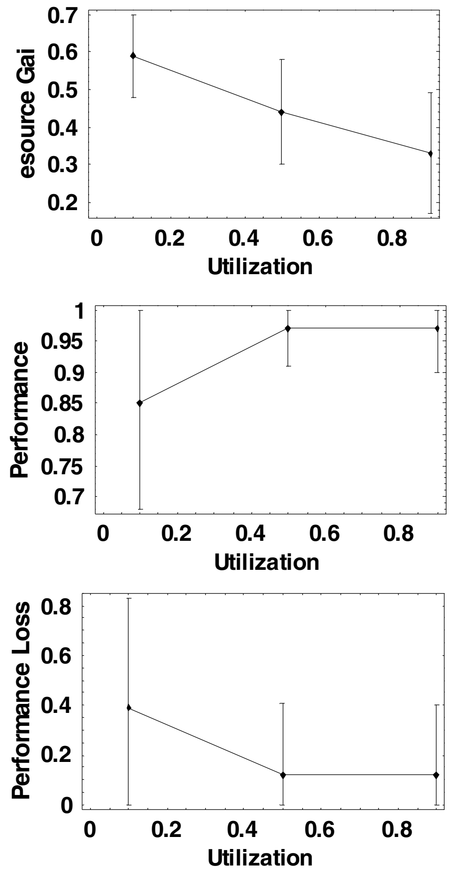


Fig. 5. Impact of Varying Utilization

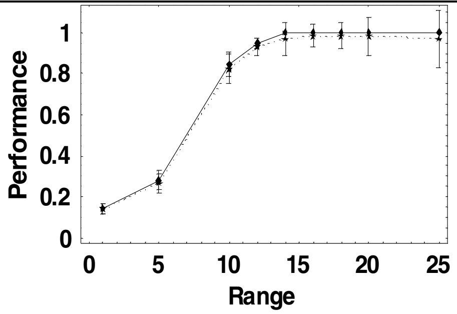


Fig. 6. Performance as Function of Range in Baseline (solid) and adaptive algorithm (dashed)

Resource gain drops with increased utilization. The higher message traffic stimulates servers to remain awake that would otherwise go to sleep, lowering the resource benefits. The system successfully adapts the number of active servers to changes in the overall message load.

This experiment is the basis for fixing utilization in subsequent experiments at 50%, a level that provides sufficient information to enable the algorithm to converge, while still making it worthwhile for servers to sleep.

4.3.3 Impact of Network Connectivity

A critical characteristic of a MANET is the range of the radios that provide the communication links. Figure 6 shows the raw performance of our scheme and of the baseline, using random node movement. We hold all parameters at their default settings and vary NodeRadius. As expected, performance increases monotonically with radio range. Importantly, the performance of our adaptive algorithm is indistinguishable from the baseline.

We have found that the directed movement of servers toward selected clients is not effective as currently implemented, as shown in Figure 7. More realistic movement models, suggested below, might yield a different outcome. We do not report further results with directed movement.

While the performance is comparable between our mechanism and the baseline, resource gain is not (Figure 8; by definition, gain for the baseline is 0). Clearly, our mechanism improves resource utilization significantly without impacting performance, compared with a best-case solution that may not be implementable.

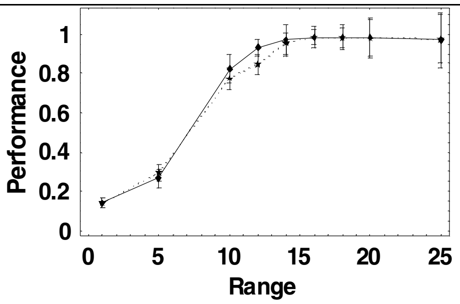


Fig. 7. Performance as Function of Range, with (solid) and without (dashed) Location Learning

4.3.4 Impact of Network Dynamics

Figure 9 shows how resource gain, raw performance, and performance loss vary as a function of server dynamics. Utilization is set at 0.5 and range at 15. For each metric, the figure shows four cases.

p_d = 0.1, p_u = 0.9—This configuration reflects highly reliable servers that seldom go down and are quickly repaired, a “best case” scenario from the operational point of view.

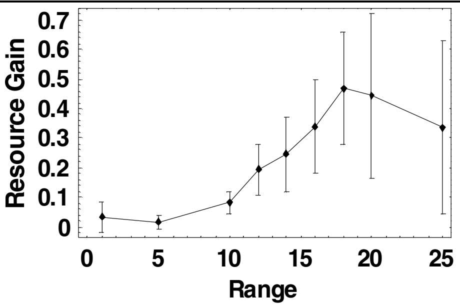


Fig. 8. Resource Gain as Function of Range

$p_d = 0.9$, $p_u = 0.1$—This configuration reflects highly unstable servers that take a long time to repair, a "worst case" scenario.

$p_d = p_u = 0.5$—This configuration reflects symmetric mean-time-to-failure (MTTF) and mean-time-to-repair (MTTR) with a moderate value.

$p_d = p_u = 0.1$—This configuration reflects symmetric MTTF and MTTR with a low value.

Consider first performance and performance loss. As might be expected, performance is good in the best case, bad in the worst case, and intermediate with symmetric MTTF and MTTR. Interestingly, performance is not significantly different between the two symmetric cases. The mean values of performance loss follow the same general trend, though wide variances make the differences less significant. Performance loss is least in the best case, when the system can reliably learn which servers to employ.

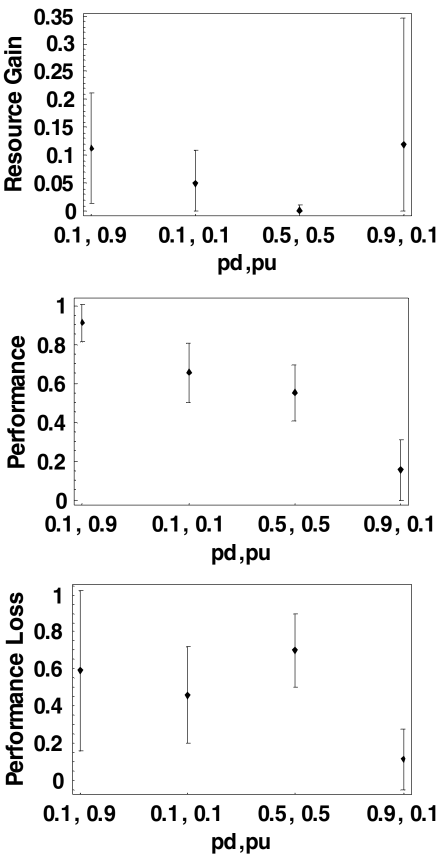


Fig. 9. Impact of Server Dynamics

Our algorithm shows resource gain in all configurations, though with high variances in both worst and best case conditions. (It is important to recognize that wide variances that reach 0 do not mean that the benefit is not statistically significant. Resource gain for the base case is identically zero by definition. Any resource gain produced by the adaptive algorithm is a real benefit, since it reflects power savings. The high variance simply means that the variation in this savings from one cycle to another is subject to wide swings, but the integral over these swings, reflecting total power saved, is unambiguously positive.) The mean resource gain in these two cases is almost the same, reflecting the benefits of adaptivity in coping with unstable systems.

In the case of equal and moderate failure and recover probabilities, there is little resource gain over the baseline. This configuration changes so frequently that our learning process does not have time to adapt to the changed environment.

5 Comparison with Previous Research

Our system addresses all three aspects of the server management problem: given the current topology of the network determined by node locations, communications ranges and node availability, decide

1. which server nodes should actually expend battery power to execute the server process;
2. to which server node a particular client should send its next service request; and
3. where to relocate server nodes to meet the current demand by the clients.

MANET's are an active area of current research, but until recently the focus of the MANET community has been on issues such as routing [9], access control [6], and security [19]. These are important issues, but largely orthogonal to the question of server management.

Recent research considers one aspect of the server management problem in MANET's, the second of our three questions (known as the service discovery problem). Efforts in this area can be divided into two groups.

Our approach is most similar to decentralized techniques such as flooding, swamping, and name-dropping (usefully reviewed in [7]), which all involve sharing knowledge of accessible services among adjacent nodes. The novelty of our approach lies in the use and propagation, not only of pointers to servers, but of scorecards to guide in selecting the server that will be tried on a given attempt. The probabilistic nature of our selection process adds robustness in the face of dynamic change. Conventional sharing schemes explore such options as whether to share with all neighbors or only with a subset at each cycle, and these options are reasonable enhancements to explore with our mechanisms.

More recent work on service discovery, and that devoted specifically to MANET's, uses service brokers to maintain directories of available servers [5, 8, 10, 18]. Highly dynamic environments (such as those encountered in military applications) can frustrate directory-based schemes.

In addition to providing a robust decentralized solution to the widely studied service discovery mechanism, our approach offers an integrated solution to the less explored problems of server activation and location. By addressing all three problems with a single set of mechanisms, we reduce the complexity of the overall system and facilitate making necessary trade-offs against different operating options, compared with approaches that piece together independent solutions to each problem.

6 Discussion and Conclusion

Swarming fine-grained agents offer an effective approach to real-time control of mobile ad-hoc networks. Our experiments show that we can reduce the resource requirements for servers in a MANET without significantly diminishing the system's performance, relative to an optimistic and probably unachievable baseline. Our experiments suggest two guidelines for when such approaches are applicable.

1. Because we rely on feedback from client attempts to access service as our source of information about the environment, the system requires a reasonable level of utilization. It is not appropriate for systems that are rarely utilized, but that must work appropriately when they are occasionally activated. However, the algorithms do adapt appropriately over a wide range of utilization levels.
2. Our methods work well when either failure probability or repair probability is low, since these characteristics lead to fairly stable server populations. When the probabilities of server failure and server repair are both high, the world changes too

rapidly for our agents' pheromone learning mechanisms, and system efficiency (as measured by resource gain) suffers.

The system described here is a highly simplified initial model of the MANET domain. We hope to explore several extensions of this domain.

- This model assumes that the movements of all vehicles are equally constrained by the same movement policy, either random (to simulate mission movement) or directed (to improve communications effectiveness). Using task allocation mechanisms similar to those we explored in [13], it would be interesting to examine fleets in which different platforms follow different movement policies, enabling some platforms learn to specialize as communication relays, and leaving other platforms more latitude for their mission-oriented tasks.
- It will also be important to examine the effect of more realistic models of mission-related movement, instead of the surrogate of random motion used here. For example, we might explore space-filling behavior to model exploratory missions, or divergence and reforming of the fleet as it moves in a general geographical direction.
- The preliminary results reported here do not show any benefit to directed movement of servers with respect to their emerging client populations. This result is counter-intuitive, and we wish to do further analysis and experimentation to understand whether and under what circumstances servers can improve system performance by directed movement.
- The breakdown of our system at low utilization levels may be mitigated in part if we make use of the "heartbeat" signals that communication nodes routinely exchange to monitor their connectivity, and we wish to explore ways that these signals can contribute to the service provider problem.
- Service provision is only one of many functions that a MANET can provide. We believe our mechanisms hold far more general promise, and look forward to expanding them into a general scheme for MANET management.

Using self-organization and emergence to engineer system-level functionality may be advantageous in many application domains, but often it is not obvious how to design the underlying processes to achieve the desired function. We discuss this aspect of the problem elsewhere [3].

Acknowledgments

This work is supported in part by the DARPA Knowledge-Plane seedling study, contract N00014-03-M-0252 to Altarum, under DARPA PM Christopher Ramming. The views and conclusions in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

References

[1] G. Aggelou. Mobile Ad Hoc Network (MANET) Papers. vol. 2004, pages Web page, University of Surrey, Guildford, UK, 1999. Available at http://www.ee.surrey.ac.uk/ Personal/G.Aggelou/MANET_PUBLICATIONS.html.

[2] S. Brueckner. *Return from the Ant: Synthetic Ecosystems for Manufacturing Control.* Dr.rer.nat. Thesis at Humboldt University Berlin, Department of Computer Science, 2000. Available at http://dochost.rz.hu-berlin.de/dissertationen/brueckner-sven-2000-06-21/PDF/Brueckner.pdf.

[3] S. Brueckner and H. V. D. Parunak. Self-Organizing MANET Management. In *Proceedings of Workshop on Engineering Self-Organising Agents (AAMAS 2003)*, pages (in press), Springer, 2003.

[4] S. A. Brueckner and H. V. D. Parunak. Swarming Agents for Distributed Pattern Detection and Classification. In *Proceedings of Workshop on Ubiquitous Computing, AAMAS 2002*, 2002. Available at http://www.altarum.net/~vparunak/PatternDetection01.pdf.

[5] L. Cheng. Service Advertisement and Discovery in Mobile Ad hoc Networks. In *Proceedings of Workshop on Ad hoc Communications and Collaboration in Ubiquitous Computing Environments (ACM CSCW 2002)*, ACM, 2002. Available at http://www.cs.uoregon.edu/research/wearables/cscw2002ws/papers/Cheng.pdf.

[6] Z. J. Haas, J. Deng, and S. Tabrizi. Collision-Free Medium Access Control Scheme for Ad-Hoc Networks. In *Proceedings of IEEE MILCOM'99*, IEEE, 1999. Available at http://wnl.ece.cornell.edu/Publications/milcom99_mac.ps.

[7] M. Harchol-Balter, T. Leighton, and D. Lewin. Resource Discovery in Distributed Networks. In *Proceedings of 18th Annual ACM-SIGACT/SIGOPS Symposium on Principles of Distributed Computing*, pages 229-238, ACM, 1999.

[8] U. C. Kozat and L. Tassiulas. Network Layer Support for Service Discovery in Mobile Ad Hoc Networks. In *Proceedings of IEEE INFOCOM 2003*, IEEE, 2003. Available at http://www.ieee-infocom.org/2003/papers/48_02.PDF.

[9] S. Lee, W. Su, J. Hsu, M. Gerla, and R. Bagrodia. A performance comparison study of ad hoc wireless multicast protocols. In *Proceedings of IEEE Infocom'2000*, pages 565–574, IEEE, 2000.

[10] J. Liu, Q. Zhang, W. Zhu, and B. Li. Service Locating for Large-Scale Mobile Ad Hoc Network. *International Journal of Wireless Information Networks*, 10(1 (January)):33-40, 2003. Available at http://research.microsoft.com/asia/dload_files/group/wireless/2002p/IJWIN.pdf.

[11] H. V. D. Parunak. 'Go to the Ant': Engineering Principles from Natural Agent Systems. *Annals of Operations Research*, 75:69-101, 1997. Available at http://www.altarum.net/~vparunak/gotoant.pdf.

[12] H. V. D. Parunak. Making Swarming Happen. In *Proceedings of Swarming and Network-Enabled C4ISR*, ASD C3I, 2003. Available at http://www.altarum.net/~vparunak/MSH03.pdf.

[13] H. V. D. Parunak and S. Brueckner. Swarming Coordination of Multiple UAV's for Collaborative Sensing. In *Proceedings of Second AIAA "Unmanned Unlimited" Systems, Technologies, and Operations Conference*, AIAA, 2003. Available at http://www. altarum.net/~vparunak/AIAA03.pdf.

[14] H. V. D. Parunak, S. Brueckner, R. Matthews, and J. Sauter. How to Calm Hyperactive Agents. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS 2003)*, pages 1092-1093, 2003. Available at http://www.altarum.net/~vparunak/AAMAS03Ritalin.pdf.

[15] H. V. D. Parunak and S. A. Brueckner. Engineering Swarming Systems. In F. Bergenti, M.-P. Gleizes, and F. Zambonelli, Editors, *Methodologies and Software Engineering for Agent Systems*, pages (forthcoming). Kluwer, 2004. Available at http://www.altarum.net/~vparunak/MSEAS03.pdf.

[16] H. V. D. Parunak, M. Purcell, and R. O'Connell. Digital Pheromones for Autonomous Coordination of Swarming UAV's. In *Proceedings of First AIAA Unmanned Aerospace Vehicles, Systems,Technologies, and Operations Conference*, AIAA, 2002. Available at www.altarum.net/~vparunak/AIAA02.pdf.

[17] P. Weinstein, H. V. D. Parunak, P. Chiusano, and S. Brueckner. Agents Swarming in Semantic Spaces to Corroborate Hypotheses. In *Proceedings of AAMAS 2004*, pages (forthcoming), 2004. Available at http://www.altarum.net/~vparunak/ AAMAS04AntCAFE.pdf.

[18] J. Wu and M. Zitterbart. Service Awareness and its Challenges in Mobile Ad Hoc Networks. In *Proceedings of Workshop der Informatik 2001: Mobile Communication over Wireless LAN*, 2001. Available at http://www.iponair.de/publications/Wu- Informatik01.pdf.

[19] L. Zhou and Z. J. Haas. Securing Ad Hoc Networks. *IEEE Network Magazine*, 13(6 (November-December)), 1999. Available at http://wnl.ece.cornell.edu/Publications/ network99.ps.

Emergent Forecasting Using a Stigmergy Approach in Manufacturing Coordination and Control

Hadeli Karuna, Paul Valckenaers, Bart Saint-Germain, Paul Verstraete, Constantin Bala Zamfirescu, and Hendrik Van Brussel

PMA Division, Katholieke Universiteit Leuven, Celestijnenlaan 300B, B-3001 Heverlee-Leuven, Belgium
{Hadeli.Karuna, Paul.Valckenaers, Bart.SaintGermain, Paul.Verstraete, Hendrik.VanBrussel}@mech.kuleuven.ac.be
zbc@acm.org

Abstract. This paper presents the design of new manufacturing coordination and control systems based on multi-agent technology. This design aims to cope with a dynamic environment characteristic for manufacturing systems nowadays. One important feature to handle these dynamics is having the ability to plan ahead, thus avoiding problems before they occur. Therefore, one novel characteristic of the system is the ability to perform emergent forecasting. Regarding emergent forecasting, an important issue that rises from this design is how to ensure that the forecast is reliable, and on the other hand, that the system is still fast enough to react against disturbances. This paper elaborates on the agents that form the system, and proposes a way to engineer it. Moreover, this paper also describes emergent forecasting. In addition to that, the trade off between responsiveness and forecast reliability (system nervousness issue) is also discussed in this paper, altogether with an example on the design of social acceptable behaviour that aims to handle the nervousness issue. Finally, some implementation and prototyping results are presented.

1 Introduction

This paper presents research on emergent forecasting in manufacturing coordination and control systems. The research team applies its approach to manufacturing control, however the applicability of the approach is broader. Manufacturing systems nowadays have to cope with an extremely dynamic environment – new products, short product life cycles, resource breakdowns, rush orders, etc. Traditional centralised, hierarchical manufacturing control architectures and their top-down development cannot cope with this increased rate of changes in manufacturing. Therefore, it is desirable to engineer a system that has the ability to self-organise. The concept of a multi-agent system offers a promising solution, consisting a set of non-centralized and mutually co-operative elements – called agents – that act autonomously, which is expected to cope with these system dynamics and uncertainties.

In this context, the real-time forecasting or prediction of the system behaviour is a valuable method to cope with the dynamics and optimise system behaviour. The novel manufacturing coordination and control system, discussed in this paper, is able to forecast emergently, thus allowing to foresee the situation on the shop floor in the near future. By having this information, the agents are more able to plan ahead. Importantly, these forecasts emerge out of the interactions of the agents.

A research prototype has been developed to test this new system design [1, 2] and it shows that this system is able to perform a load forecast for every resource in the system; moreover, for every order that enters the system, the performance of a decision strategy can be predicted. Furthermore, each order is able to react to disturbances and changes in the system, and to adapt its strategy based on the forecasts. Nevertheless, there is a trade-off between the ability of the system to react against disturbances and the ability to perform and use an accurate forecast. On one hand, when the system reacts too eagerly to disturbances, the forecast becomes unreliable and inaccurate; moreover, the system is likely to become nervous. On the other hand, being insensitive to disturbances implies that the system fails to adapt to the prevailing situation. Hence, a kind of socially acceptable behaviour must be imposed to achieve a proper balance between predict-ability and adapt-ability. This paper elaborates on the emergent forecasting and the underlying issues (handling system nervousness). Details on the basic design issues can be found in [1, 2, 3, 4].

The organization of this paper is as follows: the following section briefly introduces self-organisation. Next, the issue of emergent forecasting is elaborated, and the following chapter discusses the system engineering aspects. Subsequently, socially acceptable behaviour is discussed, along with example of the implementation. Finally, a simple prototype implementation is presented together with its result. This paper will conclude with a discussion and future works.

2 Self-Organisation

Self-organisation is described as [5]: a set of dynamical mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components. The rules specifying the interactions among the system's constituent units are executed on the basis of purely local information, without reference to the global pattern, which is an emergent property of the system rather than a property imposed upon the system by an external ordering influence. The development of multi-agent systems in this research conforms to the above definition. Agents correspond to entities in the system and lack direct specific knowledge about other entities.

A self-organising application basically is an application running without central control in which the different behaviours of the individual entities lead to an emergent coherent result [6]. This kind of applications is usually inspired on biology or physical world, chemistry, or even social systems. Characteristics of these applications are their ability to accomplish complex collective tasks with simple individual behaviours, without any central control or hierarchical structure. In this system, the individual behaviours are kept simple, in the sense that they limit the exposure of the individual entities to the overall system properties. However, the generic design for the manufacturing control systems allows these entities to become experts and to be very intelligent within their own scope. The limited exposure of each entity in the system and indirect interaction between entities enables the system to be easily implemented and the size of the system will not become a constraint. It can be implemented either in a large scale manufacturing system or small scale manufacturing system.

3 Emergent Forecasting

In manufacturing systems, determining what will happen in the future to make good decisions is a problem that must be faced quite often. Having forecast information is helpful in making a good planning which is supposed to become a good decision [7]. Therefore, the design of this new system design is able to provide this type of information. This section presents the definition of emergent forecasting and its related aspects in the manufacturing systems.

Forecasting is defined as an action to predict a future event or to estimate something in advance. Emergent comes from the word "emerge" that means to become apparent, come to light; to turn up, present itself, to appear as a result or to emanate. A system exhibits emergence when there are coherent emergents at the macro-level that dynamically arise from the interactions between the parts at the micro-level. Such emergent are novel with respect to the individual parts of the system. [8].

Hence, *emergent forecasting* is defined as *the ability to foresee what is going to happen in the near future whereby the way in which the forecast appears is not predefined, but – emergently – appears as a result of local interaction of the agents in the system.* Thus, the forecasting in this new system is performed in an emergent manner.

To enable emergent forecasting in this system, order agents, via their exploring ant agents and intention ant agents, have to interact with resource agents (see further). In addition, an order agent itself has to interact with its product agent. Exploring ant agent is an agent who is responsible to explore the possible route(s) to finish for the type of order it represents. The exploring ant agent only has local knowledge on itself; it has no idea what the other agents are or do, except for its own concerns. Next, the intention propagation ant agent is an agent who is responsible to propagate intention of the order agent. It follows the defined route and makes a reservation at the selected resources that are likely to be visited by the order along its lifecycle. Likewise, a resource agent — the one who receives intentions from order agents — only needs to handle its own business. The resource agent does not have to know what the order does before and after being served. When seen in isolation, the behaviours and action of the individual agents do not explain how a system-wide forecast is constructed; the agents only have a local concern and no sign of emergent forecasting can be seen when viewing them individually. Nonetheless, when viewing them as a whole, all of these interactions enable the forecast to emerge. This emergent forecasting can be seen at least from two perspectives, namely:

1. *From the order agent perspective.* To the order agent, the forecast provides information about the resources it is likely to visit together with the other relevant information, for instance the operations, processing times, arrival times, etc. Having this information enables the order agent to create a good plan to fulfil its goal.
2. *From the resource agent perspective.* To the resource agent, emergent forecasting provides information on future loads for the resource in the manufacturing system

that is represented by this resource agent. Resource agents are unable to accurately forecast their future behaviour without the knowledge about their future loading by the orders in the factory. To this end, orders that enter the system create order agents that express their intentions to the resource(s) that they are likely to visit during their lifecycle in the system. The detailed explanation on how the system is engineered and operates is explained in the next section.

4 The Engineering of the System

This section first describes basic guidelines that are used to design the overall system. Next, it describes types of agents in the system. Further on, the control mechanisms of this manufacturing system are described.

As a first step, the agents of the system must be identified and designed. While designing the agents, there are guidelines to be followed. Summarizing [9, 10], *agents must correspond to relevant entities* in the underlying system, and the functionality needed to answer the user requirements is implemented on top of this reflection of the world of interest.

Since an agent in this system is designed to have only local expertise, when it comes to solve problems, there is a possibility that the agent does not have enough knowledge to solve its problems. Hence, contribution of knowledge from other agents is needed. Therefore, one agent usually needs to cooperate with the other agents. To establish the cooperation between agents, agents need to perform coordination. There are two basic ways to perform coordination between agents: *coordination by direct communication* and *indirect communication*. In this research, coordination between agents employs indirect communication that is done by dropping pheromone-like information on an information board attached to each resource agent.

A pheromone is a chemical substance that is dropped by a member of natural species in the environment as guidance for other members. It is the way ant colonies propagate information while foraging for food. Biologists call it stigmergy [11]. Stigmergy describes the use of asynchronous interaction and information exchange between agents mediated by an "active" environment. One example of natural species utilizing this mechanism is an ant. The interaction of ants is based on the existence of a smelling chemical substance called a pheromone. Ants deposit pheromones in their environment, which are observed by other ants and this influences their behaviour. The pheromone depositing mechanism supports two major operations mediated by the environment in which the insects are situated, namely aggregation (accumulation of information) and evaporation (disappearing of old and expired information). Attractive properties of this ant mechanism are:

1. Evaporation makes the colony forget the information that is no longer valid. Stale information automatically disappears with time, as it is no longer refreshed.
2. Environment is reused in the solution (no maps in the brain of the ants).

From these examples in nature, the following principles are derived [4]:

1. Make the environment part of the solution to handle a complex environment without being exposed to the complexity of this environment. Note that this also complies with the essential modelling approach in object-oriented design.

2. Deposit relevant information in this environment, ensuring that locally available data informs about remote system properties, supporting system-wide coordination.
3. Limit the lifetime of this information and refresh the information as long as it remains valid. This is a "forget and refresh" mechanism, which is a basic mechanism to handle dynamics in systems.

4.1 The Agents

In this research, the manufacturing control system implements the PROSA reference architecture [12]. According to PROSA, there are three basic agent types needed to construct a manufacturing control system: order, resource and product agents.

An *order agent* represents a task in a manufacturing system. It has a responsibility to ensure that all the assigned work is properly executed. It manages the physical product instance being produced. Its knowledge space is limited to the order information and state of the workpiece: e.g. due date, order state model, etc. The order agent knows nothing about the other orders. An order agent is connected to each workpiece (or a group of workpieces) throughout its life cycle.

A *product agent* holds the process and product type knowledge. It provides information on how to make instances of its product type correctly (mainly to order agents). A product agent maintains consistent and up-to-date information on the product design, process plans, bill of materials, quality assurance procedures, etc.

A *resource agent* reflects a physical entity, namely a production resource of the manufacturing system, and an information processing part that controls the entity. It offers knowledge about production capacity and functionality to the surrounding agents. Its knowledge space is limited to self-monitoring and self-control. It also has a list of resources that connect directly to it.

These three basic agents are structured using object-oriented concepts like aggregation and specialization. Moreover, these agents are equipped with an information board on which other agents can put, observe and modify information. The information on a board has a finite lifespan; information disappears after certain amount of time. A blackboard is attached directly to an agent. Thus, it is local and only contains information that has relation to the corresponding agent (analogous to the clipboard attached to the hospital beds in old movies). The evaporation mechanism makes it different from normal blackboard implementation [13]. The next section discusses the multi-agent coordination mechanisms.

4.2 Control Mechanisms

When an order is dispatched to the shop floor, it has to discover a suitable route to get itself produced; note that predefined routings intrinsically are unsuited for dynamic manufacturing environments. In the control systems in this paper, order agents are responsible for this task. Order agents are linked to their physical product instance(s).

To find a proper route through the system, the order agents create and send out mobile agents (called ants in this manuscript) that travel virtually through the network of manufacturing resources and find possible solutions. To accomplish this task, there are two activities to be carried out, namely *an exploration activity* and *an intention* *propagation activity*. During their lifecycles, the two types of ants corresponding to these activities, deposit pheromones in the environment. Based on the observation of the pheromone information, order agents decisions are influenced. The following section will explore more detail on these two activities.

4.2.1 Exploration Activity

Order agents represent the physical orders in the system. An order agent has the responsibility to ensure that its product instance is produced in time, undergoes all necessary operations, and reaches certain quality level. Hence, the order agent has to define the routes (solutions), which its product instances can follow. A solution is defined by its routing information together with the resources, operations, starting time, queuing time, etc. The shop floor consists of a network of resources. To explore the available possibilities, the order agent creates mobile agents called *exploring ant agents* that inherit their problem solving behaviours from their creator. The exploring ant agent virtually travels through this network, starting from the position where the work piece and order agent reside until the exit. To accomplish its task, the exploring ant agent observes the local blackboards attached to the resource agents and uses the pheromone information deposited there as a search guideline. The strategy to explore the network is a plug-in[1] for the control system. Not every exploring ant agent uses the same strategy. Some of them look for the promising routes whilst the other will look for the route that avoids critical resources.

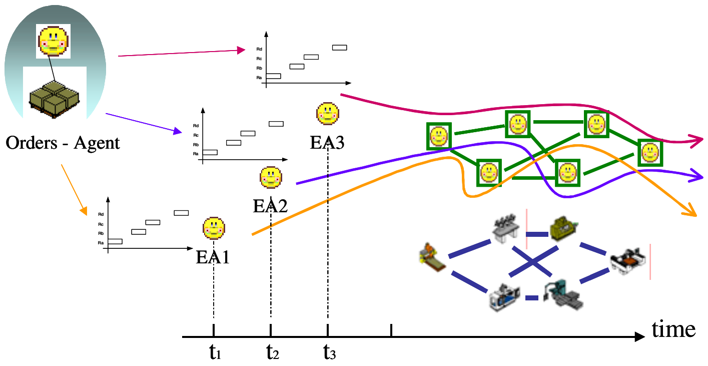


Fig. 1. Exploring ant's task

Exploring ants are created when needed. Each exploring ant explores the network, travels on different possible route and records necessary information, for instance: the resource(s) it has visited, the estimated starting time at every resource(s), waiting time, duration of each processing step, the cost of operation. The recorded information is useful to predict the travel time and the overall processing time. Moreover, it is used to determine the quality of the solution. After performing the task, the exploring ants report back their findings (solutions) to the order agents. The result is represented

[1] A naturally fitting possibility to implement such a plug-in is a design following [14].

as a list of operations together with the resources where it will be processed, starting time at the resources and the duration. The mechanism can be seen in Fig 1.

Exploring ants are created at a suitable frequency along the lifecycle of the order, and each of them investigates the expected performance of a single feasible route of the product instance through the factory. Note that the resource agents, on which the pheromones are deposited are more cooperative and intelligent than the stones, earth and foliage on which the real ants deposit their pheromones. The resource agents possess a kind of *reservations department* (like a hotel or airline) that answers queries about future availability. This reservations department is able to give accurate answers because intention ants make bookings on behalf of the order allowing the resources to self-schedule their work into the near future (based on self-knowledge).

4.2.2 Intention Propagation Activity

To get services from resources, order agents express their intention; this intention is a sort of temporary commitment to the set of affected resources. Based on the information provided by exploring ants (i.e. solutions collected by these ants), the order agent selects one solution, which is considered to be the best, to become its intention. Next, the order agent propagates this intention to the resources it intends to visit. To do this, the order agent creates and sends out mobile agents called *intention ants* at a suitable frequency. The intention ants virtually execute the remaining routing and processing steps of the selected candidate solution, visiting all resources that are listed in the selected solution. Figure 2 shows the above-mentioned mechanism. At each resource, this ant expresses the intention of its order agent by requesting a time slot to be reserved in this resource.

An intention ant informs the resource agent that its workpiece is likely to be processed on that resource and will arrive at particular point of time. As a response, resource agent looks into its diary book, marks a time slot for the particular order, and returns the updated performance information, namely possible starting time, duration of operation, and waiting time information to the ant. In this case, the intention ants enable an emergent forecasting of the resource utilization. During its journey, the intention ant collects and updates the time and status information from the resource agents it has visited. Therefore, any changes and disturbances – resource breakdown, rush orders taking your slot – instantly become visible during refresh. Slot reservations evaporate and disappear unless refreshed. Therefore, an order agent frequently propagates its intention to have its workpiece processed at particular resources – i.e. reconfirmation of current reservations.

When refreshing an intention, the performance of the current intention is re-evaluated. When the expected performance deteriorates during such refresh or when an exploring ant detects a better alternative, the order agent will have a tendency to shift to such better alternatives. If an order agent always immediately changes to a new intention when it perceives a better alternative, the system is likely to become overly nervous and the load predictions, which rely on the intentions to represent some level of commitment from the order agents, become inaccurate and the forecast information will be useless. If all order agents react immediately on the smallest perceived advantage, the system probably becomes unstable/chaotic. Therefore, this behaviour should be dampened; an order agent must not change its intention too easily. In the remainder of this paper, this phenomenon is called *system nervousness* and is defined as the frequency of intention changes by the current orders.

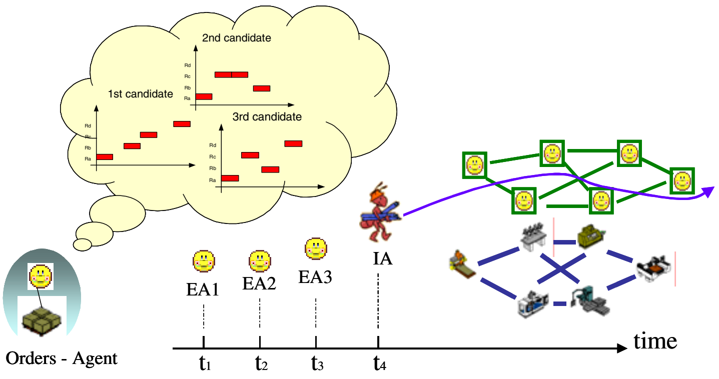


Fig. 2. Intention ant's task

This situation actually is a trade-off situation. If the order agents fail to account for their perception of the prevailing situation, the system fails to adapt and suffers from poor performance under dynamic conditions. However, if the order agents are extremely eager to change, then they fail to uphold their commitments and the forecasts will be useless; the overall system may become too nervous and instable. This will cause confusion on the shop floor; the system will become costly due to frequent rescheduling and also fluctuation in capacity utilization [15]. To overcome this problem, order agents must have a kind of *socially acceptable behaviour* to balance between the two extremes (being unaware and being too nervous). The following section elaborates one implementation of the socially acceptable behaviour. Remark that the described algorithm is only one out of many possible solutions.

5 Socially Acceptable Behaviour

To shift from the current intention to a new intention, order agents must ensure that the new solution is better than the current intention. The word "*better*" here indicates that the new solution has a better performance relative to the current intention. There are several criteria to define a better solution. In this application, implemented on the top of the discussed framework, the order lead-time is chosen as a performance measurement. The overall idea of the socially acceptable behaviour is not limited to the use of lead-time; it can be extended and customised if necessary.

Lead-time itself is the length of time needed to deliver a product from ordering till acceptance by the customer [16]. In this paper, a more specific definition of lead-time is used, called order lead times. Order lead times is a sum of the individual operation lead times at all work centres passed by this order [17]. A new solution can be considered as a "better solution" if it shows a decreasing in its order lead-times. To measure changes, a scale from 0 to 1 is used and is called significance level in this paper. The larger the significance level specifies that the new solution is empirically better than the current intention. The following explanation will describe how to come up calculate the significance level.

Order lead-time can be defined as

$$OLT = t_{OC} - t_{OR} \tag{1}$$

where:
OLT : Order Lead Time
t_{OC} : order completion time
t_{OR} : order released time

Order lead-time can also be calculated from the sum of the operation lead times:

$$OLT = TL_1 + TL_2 + ... + TL_k \text{ or } OLT = \sum_{i=1}^{k} TLi \tag{2}$$

where:
TL : lead time of one operation

Order lead-time of one operation (TL) is composed of:
1. Operation time (t_o)
2. Waiting time (t_w)

More reward points are allocated when the new solution shows a tremendous reduction in waiting time. Nonetheless, this is not mandatory. The value of the rewards can follow any function (plug-in) that is defined by the order agent. Furthermore, changing intentions brings a cost to the whole system. The benefit of changing intention has to be high enough to justify this cost. By using an exponential function (see below), small changes in intention performance have a far less chance of causing an intention change of the order agent.

The function parameters can be used to adjust the function slope. Furthermore, they can be changed from time to time. It mostly depends on the way the order agent observes the shop floor condition. When the shop floor is very dynamic, the function parameters can be set to dampen the system (agents ignore all but major disturbances), whereas the parameters can be adjusted to make agents more responsive if the overall behaviour remains stable. Through this mechanism, the order agent not only can adapt to the disturbances happening on the shop floor, but also can adapt to the rate by which these disturbances occur. The function itself can be represented as:

$$f(x) = n^x;\ 1 \le n \le \infty \tag{3}$$

where:
n : parameter to change the slope of the curve, defined by the order agent.
x : deviation of total waiting time or deviation of total operation time.

In this function, n is a tuning parameter. The reward function can vary over different implementations. More details concerning this function are out of the scope of this paper.

Depending whether the order agent has propagated an intention or not, it will show a different behaviour as described in Table 1. Secondly, an order agent must evaluate the quality of the new solution compare to the current intention. Note that an order agent always refreshes its last intention before comparing it with the new solutions.

Table 1. Treatment between has propagate intention or not

No intention ever propagated yet	**Has propagated any intentions**
1. Wait for several exploring ant agents to return and provide the exploration result. 2. Build a set of results. 3. Select one result with the smallest order lead-time and set as current intention.	1. Wait for several exploring ant agents to return and provide the exploration result. 2. Build the competitor set of results. 3. From this set of result, select the best performance solution, i.e. the one with the smallest lead-time. 4. Compare the new solution with current intention. More details are described below.

In order to measure the quality of the new solution, it is necessary to measure how significant is the changes compare to the current intention. The significance level is composed of two elements, namely significance level from the waiting time and significance level from the operation time. To measure the significance level, the following steps are executed.

1. Calculate the current intention's waiting time, $\sum t_w$.
2. Calculate the new solution's waiting time, $\sum t_w'$.
3. Find the differences between those waiting times.

$$\Delta t_w = \sum t_w' - \sum t_w \tag{4}$$

4. Calculate the reward points from this difference.

$$R_{\Delta t_w} = f\left(x = \Delta t_w\right) = n^x \tag{5}$$

5. The contribution from waiting time to the overall significance level (S_{t_w}) is calculated as follows:

- Multiply the differences between the waiting times and the reward points.
- Divide the previous result by the maximum value of this contribution, which reflect to the worst result reported back by the exploring ants.
  The maximum value of this contribution is the multiplication between the maximum differences between the waiting times, which in this paper is defined to be equal to the current intention's waiting time, and the reward points from this maximum difference. The following equations are the reflection of this step.

$$\Delta t_{wMAX} = \sum t_w \tag{6}$$

$$R_{\Delta t_w MAX} = f\left(x = \Delta t_{wMAX}\right) = n^x \tag{7}$$

$$S_{t_w} = \frac{\Delta t_w \times R_{\Delta t_w}}{\Delta t_{wMAX} \times R_{\Delta t_w MAX}}; S_{t_w} \in \{0,1\} \tag{8}$$

$$\text{if } S_{t_w} > 1, S_{t_w} = 1$$

6. The same procedures (from step 1 to step 6) are applied also while calculating the significance level contributed by the operation time. However, the reward function has different parameter values.
7. Finally, to come up with significance level, firstly sum the contribution from both aspects and divide by 2. It can be written as:

$$SL = \frac{S_{tw} + S_{to}}{2} \tag{9}$$

Significance level mirrors the level of confidence of the order agent to shift to the new solution. Thus, when the significance level is below certain threshold, the order agent will not take any risk shifting to the new solution. The order agent defines the threshold level. Again, the threshold level's value is floating, It is mostly depends on the historical data and can be changed if necessary.

Social Acceptable Behaviour and propagation intention policy

Although from the above mechanism, the order agent already has the confidence level to shift to the new solution and willing to propagate the new intention, the order agent should not change intentions easily. It should behave according to social rules as in human society where someone who already engaged in a certain appointment should not easily call off without any strong and crucial reason. Without going into a psychological analysis on this behaviour, this situation can simply be handled by at least two types of procedures, namely:

1. Applying a probabilistic mechanism each time the order agent is willing to change intention (see further).
2. Limit the frequency to change intentions. This procedure is simply sets a maximum frequency at which an order agent is allowed to change intentions.

Probabilistic mechanism applied to intention changing procedure

This mechanism can be explained as follow.

1. Order agent has its own frequency rate to refresh its intention. If no social acceptable behaviour is applied, an order agent will change its intention any time it likes.
2. At every refreshing cycle, every order agent has to draw a random number, r_i.

3. If r_i is smaller and equal to the confidence level to shift (a value between 0 – 1), then the order agent is allowed to change its intention. Otherwise, it has to keep its old intention.
4. Furthermore, no matter how big is the value of the significance level, the system should guarantee that if the significance level stays the same all the time, then at the end of a finite time horizon, this change should be announced. Consequently, when defining whether to change intention or not, the number of repetition in which the significance level is the same should be taken into account. Furthermore, the strategies to do it can be varying in implementation, for example double withdraws of random variable at every cycle, or increase the confidence level to shift from this cycle onwards.
5. The visual description is given in Fig. 3. The width and similar height bars indicate the confidence level to change at every refreshing cycle. The slim bars indicate the random number drawn by the order agent at each refreshing cycle. It is shown in the figure that if the slim bars are under the width bar, the order agent will change its intention.

By applying the above-mentioned mechanism, order agents will be able to propagate new intentions wisely. As a result, when there is disturbance on the shop floor, e.g. a machine breakdown, not all order agents that intend to be processed on that machine will switch to the other machine(s) that offer the same capability. As a consequence, the other machine(s) will not suddenly become overloaded by these orders.

Note that the above is only one possibility among many and is subject of ongoing research. The objective is to balance responsiveness against forecast reliability.

6 Implementation and Prototyping

The implementation of the overall system above, especially the part concerning social acceptable behaviour's algorithm is ongoing. Nonetheless, prototypes have been developed and the basic functions of this system have been validated. The following part will present the prototype that addresses a real industrial case.

This prototype reflects the real manufacturing plant that produces weaving machines. Some data on the case has been modified for confidentiality reasons. The plant can be described as follow: the shop is composed of 13 processing resources and 1 exit station. An automatic storage and retrieval system (ASRS) is installed in the middle of the shop and acts as storage for either finished or work-in-process (WIP) parts. Altogether, the resources and the ASRS are connected with a 'tram' that transports parts from one workstation to another workstation or ASRS. From this test bed, several results can be shown.

Emergent forecast in order agent and resource agent

A set of experiments was carried out on this prototype. Certain numbers of order are released to the shop floor. In this case study, orders that arrive on the shop floor have no predefined route to finish. Thus, an order agent has to send out exploring ant agent to discover the promising route and furthermore, utilises this information to propagate intentions to the set of resources it needs to get services from.

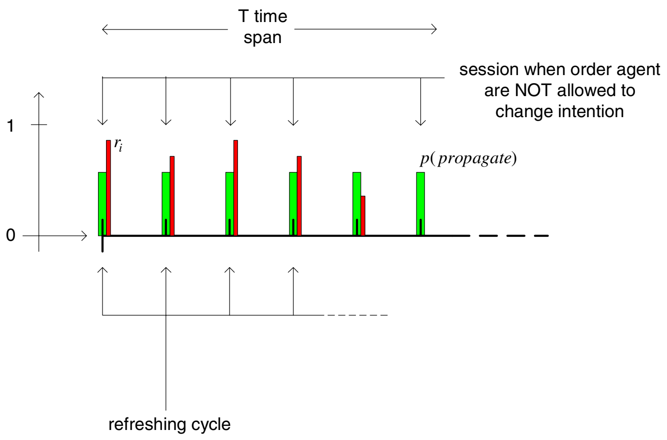

T time span
session when order agent are NOT allowed to change intention
1
0
r_i
$p(propagate)$
refreshing cycle

Fig. 3. The implementation of social acceptable behaviour

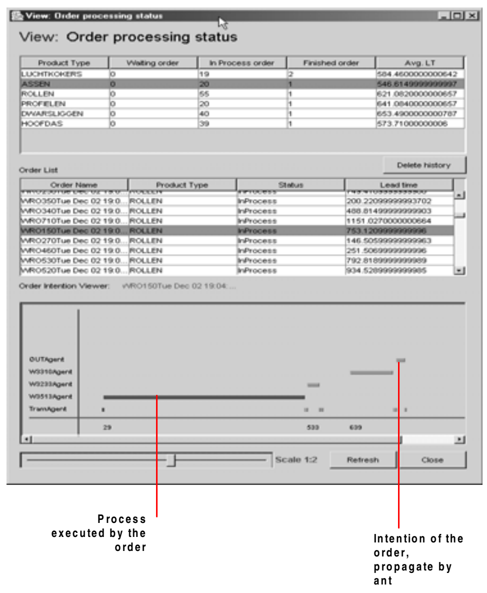
The result of this implementation is visualized in Fig. 4. Figure 4 shows the current intention of one selected order (WR0150 of type Rollen). The black bar indicates operations that have already executed, and the light grey bars indicate the remaining operations altogether with the possible start time, duration and resources that are going to process these operations. This result contains information about the set of resources that the order is likely to visit, the starting time and the duration of stay. This solution is built from the interactions between order agent (through its exploring ants and intention ants) and resource agents.

The solution presented in this panel is not a definitive solution, because as the shop floor changes from time to time, the solution may also alter. For example, take order number WR0130 of type Rollen, its early intention was to have its second last operation processed on resource W3310 at time 2034, however, since there are some disturbances (extension of operation time of current processed order in W3310) the starting time of the operation may be postponed. See Fig. 5.

At every resource agent, intentions from the order agents are recorded and translated into a kind of Gantt chart. The Gantt chart reflects the current and future load in the resource agents. As mentioned before, order agents express their intention through their intention ants. When arriving in a certain resource, the ants book a time slot for the order agents. Thus, this information actually shows the information about orders that intent to visit and request services (see Fig. 6).

The accuracy of the forecast

As mentioned in earlier chapter, order agents are sending out intention ant agents to express their intention. The intentions are changing from time to time, and are expected to be closer to the actual solution as they approach the execution time of the operations. Fig. 7 shows the error plot of the data that starts from the propagation of intention until the real execution time of the order.



View: Order processing status

Product Type	Waiting order	In Process order	Finished order	Avg. LT
LUCHTKOKERS	0	19	2	584.4600000000642
ASSEN	0	20	1	546.6149999999997
ROLLEN	0	55	1	621.0820000000657
PROFIELEN	0	20	1	641.0840000000657
DWARSLIGGEN	0	40	1	653.4900000000787
HOOFDAS	0	39	1	573.7100000000006

Order List

Order Name	Product Type	Status	Lead time
WRO350Tue Dec 02 19:0...	ROLLEN	InProcess	200.22099999993702
WRO340Tue Dec 02 19:0...	ROLLEN	InProcess	488.81499999999903
WRO710Tue Dec 02 19:0...	ROLLEN	InProcess	1151.0270000000664
WRO150Tue Dec 02 19:0...	ROLLEN	InProcess	753.1209999999996
WRO270Tue Dec 02 19:0...	ROLLEN	InProcess	146.5059999999963
WRO460Tue Dec 02 19:0...	ROLLEN	InProcess	251.5069999999996
WRO530Tue Dec 02 19:0...	ROLLEN	InProcess	792.8189999999989
WRO520Tue Dec 02 19:0...	ROLLEN	InProcess	934.5289999999985

Order Intention Viewer: WRO150Tue Dec 02 19:04: ...

Fig. 4. Emergent forecast information from order agent WR0150

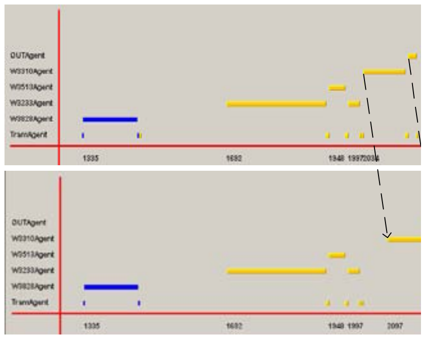


Fig. 5. Emergent forecast of order number WR0130. (i) top figure is the earlier intention (ii) bottom figure is the later intention

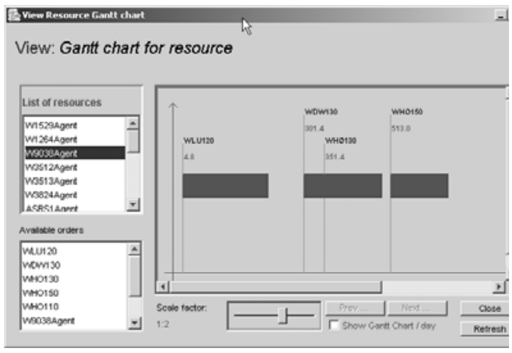


Fig. 6. Emergent forecast information at resource agent – Gantt chart at selected resource

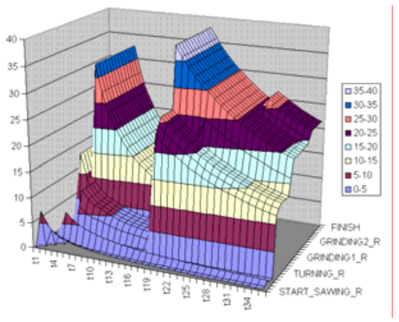


Fig. 7. Error plot of intention data of order R04

As seen in Fig. 7, when the time is approaching the real execution time, the forecast become more accurate. It can be seen from the decreasing value of error. The big slope changes in the middle of the graph are due to the down time of a machine. The systematic jump in inaccuracy, when looking further ahead in the future, is caused by about-to-be-released orders being unable to propagate their intentions before release.

7 Discussion and Future Works

This paper described how a new manufacturing coordination and control design produces an emergent forecast. Emergent forecasting is an important feature in current and future manufacturing coordination and control systems since it provides the ability to foresee the future shop floor condition. Having emergent forecasts enables the system to avoid problems before they occur and offers the possibility to optimise performance. To do so, order agent sends out two types of mobile agents – called ants. They are *exploring ants* and *intention ants*. Exploring ants are responsible for performing a scouting task throughout the network of resources to find an attractive route in order to accomplish the task of their order agents, where intention ants are responsible for notifying resources that an order agent is likely to visit in the future.

To order agents, the emergent forecasting provides information about the load of resource(s) the order agents are likely to visit in the future. It also provides information on their arrival time at each resource, how long they will remain there and some other relevant information. This mechanism enables the order agent to forecast its finishing time and also calculates its performance before it is executed. Moreover, when suddenly the order characteristic is changed, for instance from normal order to rush order, and it needs to change its priority, this forecasting information can be used as a guidance to find the solutions (routes) that will provide the earliest finishing time.

Furthermore, if some intelligent algorithm is implemented inside the order agent, and it is useful to analyse this information and behave differently at different situations, for instance when the shop floor is very dynamic, the order agent can sense this from the emergent forecast information and reacts accordingly, or the order agent can also experiment with the performance achievement by changing its behaviour.

At the resource agent, the emergent forecast enables it to foresee its near future load. This information enables the resource agent to manage its own logistical tasks, for instance, to schedule its downtime for maintenance or other non-process related activities. By having this information, resource agent can schedule its downtime, and ensure that it is not scheduled in a heavy loaded period. For example, in a factory where each machine is connected with tram and automatic storage and retrieval system, the tram can utilize this information to foresee the time slot where it has low load. At this low load time, tram can move the location of container to the position near to the destination. By doing this, the transportation time to move this container will automatically reduce. Furthermore, the resource agent can also be designed to have an intelligent decision taking mechanism, for instance, applying its own dispatching rule on the orders that queue for its services. Thus, instead of First In First Out, other dispatching rules can be applied. This behaviour enables the increase of the performance of the overall system.

All these routines that perform by these agents automatically create a kind of self-reinforcement cycle. As seen in Fig. 8, the propagation of intention will provide resource agents with the prediction on orders that require their services in the future. Having this information, resource agent(s) can do a better self-management. As a result, exploring ant agent will receive a better response/forecast while requesting information about services from the resource agent. Furthermore, if exploring ant agent can provide a better forecast to its host, then order agent can propagate a better intention. Consequently, better intention means better prediction for resource agent(s), and if this cycle is continuing on, then both order agents and resource agents will have a more accurate result.

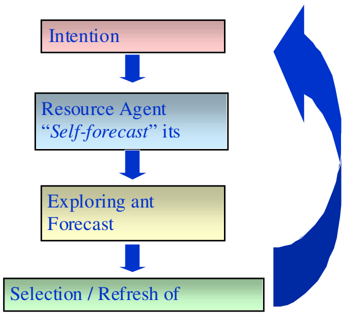


Fig. 8. Self-reinforcement cycle of this emergent forecast mechanism

The system's nervousness issue is also an important aspect that comes along with the emergent forecast capability. This issue is strongly related to the reliability of the forecast created by exploring ants. Due to the unreliable forecast, order agent tends to propagate new intention faster and more frequent. As a consequent the system will become nervous and the previous forecast will become useless. As expected, any forecasts should be reliable and its quality is measured by whether it is capable or not to predict the future events accurately. In the real application, this unreliable forecast causes the existing schedule in resource agent turn out to be invalid. Consequently, resource agent should reschedule all the waiting tasks/operations and its own logistic activities, and thus, new adjustment, new set-up, new allocation that come along with the operation should be done. As a result, the process will become more costly, and this is certainly undesirable. Furthermore, the rescheduling will then affect the performance of the other orders that are waiting to be processed on the resource(s). When the forecast is useless and unreliable, the ability to foresee the future events automatically deteriorated. Moreover, the self-reinforcement cycle will break.

Applying the social acceptable behaviour to the order agent is one way to overcome this problem. Further observation on the results is needed to verify the above-mentioned concept. Apart from those algorithms, at the system level, there are also a series of tuning parameters that influence the behaviour of the system, for instances frequency of ant propagation, evaporation rate. In current prototype, these parameters are tuned manually through user-interface. Numerous experiments are still needed in order to provide a clear report on how those parameters influence the system's behaviour. Furthermore, in the future, there is a thought to enable an auto-tuning mechanism on these parameters, so that the system can adjust itself in order to attain the better performance of the system.

The overall design seems not complete if there is no evidence on how the performance of this new system compare to the current available/practice system. Therefore, for further work, benchmarking of this idea will be performed. To support this, a benchmarking service is under development within the EU Network of Excellence on Intelligent Manufacturing Systems (cf. www.ims-noe.org).

Acknowledgement

This work was supported by the European Commission under GROWTH program and the K.U.Leuven research council (GOA-AgCo$_2$).

References

1. Hadeli, Valckenaers, P., Kollingbaum, M., Van Brussel, H., Multi-agent coordination and control using stigmergy, Computers in Industry 53 (2004) 75–96
2. Zamfirescu, C., Valckenaers, P., Hadeli, Van Brussel, H., and Saint Germain, B., A Case Study for Modular Plant Control, Lecture Notes in Artificial Intelligence, Vol. 2744, Springer-Verlag, Berlin Heidelberg New York (2004) 268-279
3. Valckenaers, P., Van Brussel, H., Kollingbaum, M., Bochmann, O., Multi-agent coordination and control using stigmergy applied to manufacturing control. Lecture Notes in Artificial Intelligence, Vol. 2086, Springer-Verlag, Berlin Heidelberg New York (2001) 317-334
4. Valckenaers, P., Saint Germain, B., Verstraete, P., Hadeli, Zamfirescu, C., Van Brussel, H.: Ant Colony Engineering in Coordination and Control: How to Engineer an Emergent Short-Term Forecasting System. Proceedings IWES'2004, Budapest (2004).
5. Parunak, H.V. D., Bruekner, S. A., Engineering Swarming Systems, Forthcoming in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, eds., Methodologies and Software Engineering for Agent Systems. Kluwer (2004)
6. ESOA WG – Mission Statement http://cui.unige.ch/~dimarzo/esoawg/mission.doc
7. Hogarth, Robin M., and Makridakis, S., Forecasting and Planning: An Evaluation, Management Science Vol 27 No. 2 (1981) 115- 138.
8. De Wolf, T, Holvoet, T., Emergence and Self-Organisation: a statement of similarities and differences. Proceedings of the International workshop on Engineering Self-Organizing Applications (ESOA) 2004, New York (2004).
9. Valckenaers, P., Van Brussel, H., Hadeli, Bochmann, O., Saint Germain, B., Zamfirescu, C., On the design of emergent systems: an investigation of integration and interoperability issues., Engineering Applications of Artificial Intelligence 16 (2003) 377-393
10. Parunak, H. V. D., Sauter, J., Clark, S., Toward the Specification and Design of Industrial Synthetic Ecosystems, Fourth International Workshop on Agent Theories, Architectures and Languages (ATAL) (1997)
11. Theraulaz, G., E. Bonabeau, A Brief History of Stigmergy, Artificial Life, 5 (1999) 97-116
12. Van Brussel, H., J. Wyns, P. Valckenaers, L. Bongaerts, P. Peeters, Reference architecture for holonic manufacturing systems: PROSA. Computers In Industry, 37 (1998)
13. Ciancarini, P., Omicini, A., and Zambonelli, F., Coordination Models for Multi-Agent Systems. AgentLink News 3, July (1999)
14. Brückner, S., Return from the Ant – Synthetic Ecosystems for Manufacturing Control. PhD Thesis, Humbold University, Berlin, Deutschland (2000)
15. Campbell, Kenneth L., Scheduling Is Not The Problem, Production and Inventory Management Vol. 12 No. 2 (1971) pp. 53-60.
16. Sipper, D., Bulfin, Jr, R.L., Production Planning, Control and Integration, McGraw-Hill International New York (1998)
17. Wiendahl, H.P., Load-Oriented Manufacturing Control, Springer-Verlag, Berlin Heidelberg New York (1994)

IDReAM: Intrusion Detection and Response Executed with Agent Mobility
The Conceptual Model Based on Self-organizing Natural Systems

Noria Foukia

University of Geneva, rue Général Dufour 24,
CH-1211 Geneva 4, Switzerland

Abstract. Nowadays, lots of researches in *Intrusion Detection* and *Intrusion Response* try to find new solutions to circumvent new intrusive behaviors. One of the principal weaknesses of these systems is the lack of robustness inherent in their centralized nature. Even though most of the existing *Intrusion Detection and Response Systems* (IDRSystems) use distributed data collection (host-based or network-based) many of them continue to perform data analysis centrally, thereby limiting scalability. Moreover, even if the IDRSystem is distributed in the network, its deployed elements generally remain static. With the means available to modern attackers, such as automated intrusion tools, these static and distributed elements are easily accessible. Often, this does not always contribute to improving the reliability and resistance to attacks of such static components.

This paper presents our approach for building an IDRSystem called *Intrusion Detection and Response extended with Agent Mobility* or IDReAM for short. IDReAM combines *Mobile Agents* (MAs) with self-organizing paradigms inspired by *natural life systems*. This approach was already announced in a preceding paper [4], and the present paper describes in a more detailed way the conceptual model. All the research works relating to IDReAM are gathered in a PhD Thesis [3] which also contains the implementation results of the model and its evaluation. The present paper is limited only to the model.

Keywords: self-organization, mobile agents, intrusion detection and response.

1 Introduction

The collective and complex behavior emerging from simple interacting entities is frequently illustrated in natural phenomena such as biological systems (body cells, human brain, etc.) or insect colonies (ants, termites, bees, etc.). These natural systems appear completely distributed and exhibit efficient characteristics in term of robustness, and are perfectly adapted to solve complex tasks. For a few years these complex natural systems have been a source of inspiration for many research projects in computer science; the scope of influence ranges from artificial intelligence with the work done by Marco Dorigo in swarm intelligence to telecommunication problems such as adaptive routing and optimal resource allocation [1] in computer networks. Beyond computer science this influence reaches other research topics such as distributed robotics systems [10] and economics. And the list is not exhaustive. One interesting point is that the population of these natural systems consists of simple moving entities or agents naturally dispatched throughout the environment. In spite of their individual simplicity, these entities present a highly structured self-organizing capacity to accomplish complex tasks. This collective and self-organizing capacity emerges from inter-entity interaction performed through the hosting environment and it exceeds the capacity of each individual.

The efficiency of natural systems as complex systems completely distributed and dynamic, was a source of foundation of IDReAM. The *Intrusion Detection System* (IDSystem) was designed with the immune system in mind whereas the *Intrusion Response System* (IRSystem) was designed with the ant colony organization in mind. The following points out what was presented in [4]:

The IDSystem borrows mechanisms from the immune system that protect the human body against external aggressions: special cells of the immune system, called the T cells, travel around the body to detect possible aggressions by eliminating the proteins that they do not recognize as safe proteins. These bad proteins are called non-self proteins in the medical jargon. *Intrusion Detection Agents* (IDAs) roam the network to detect bad suspicious behaviors mimicking the behavior of T cells.

The IRSystem borrows mechanisms from the stigmergic paradigm of a colony of ants. At the time of foraging, the ants use the environment to diffuse a chemical substance called the *pheromone* which traces the route for the other ants from the nest to the source of food. *Intrusion Response Agents* (IRAs) roam the network to respond to the IDAs' alerts mimicking the behavior of the ants to trace the route to the alert and give the response.

The two natural systems exhibit a social life by the organization of their entities (T cells and ants) which is not possible without the functionality of mobility. Thus, in a natural way, MAs are good candidates to provide this property of mobility. In practice the two populations of MAs, IDAs and IRAs, play the role of autonomous entities permanently distributed over the network and embody the self-defending and self-organizing behaviors.

There are different reasons that motivated the choice of two distinctive models for the IDSystem and IRSystem. The main reasons are:

- The behavior of T cells acting as sentinels of the human body is more adapted to the detection task. To detect an intrusion, the IDA needs to look for what happens in the system and to compare the events it encounters with normal situations. This is what T cells do in the human body.
- The behavior of ants tracing the pheromone to find the source of food is more adapted to the response task. When the intrusion happens the IRA

needs to respond at the location of the intrusion. Remotly, it needs a means to reach the suspicious location. The information of locality (pheromonal gradient in the IRA's neighborhood) provided by the pheromone appears to be a simple and natural way to also trace the route to the alert.
– Moreover, this separation of concerns between IDAs and IRAs is a major advantage because it makes IDReAM more robust since an attacker needs to understand and adapt to both populations of agents.

The rest of the paper is structured in the following way: The Section 2 gives an overview of the immune system. The Section 3 gives an overview of the ant stigmergic system. The Section 4 details the various elements of the mapping between IDReAM and the two natural systems described in the Section 2 and 3. The Section 5 describes the conceptual model of the IDSystem part and the Section 6 describes the conceptual model of the IRSystem. The Section 7 discusses the conceptual model and draws the conclusion.

2 Immune System – An Overview

The structure of the human body defense system is multi-layered with defenses provided at many levels, from the skin which is the outermost barrier of protection to the adaptive immune system which can be viewed as a distributed detection system in the body. This immune system [9] is a complex network of specialized cells and organs that has evolved to defend the body against diseases and infections by "foreign" invaders such as bacteria, viruses, fungi, parasites, and debris. In a first step the immune system attempts to prevent or stop these external organisms before they enter the body. In a second step it seeks their presence in the body in order to destroy them. For that, it distinguishes between molecules and cells of the body called "*self*" from foreign ones called "*non-self*". The body's immune defenses normally coexist peacefully with cells that carry distinctive self marker molecules. But when the immune defenders encounter non-selfs they have to eliminate them quickly. Any substance which is capable of causing an immune alert is called an *antigen*. An antigen is recognized by means of markers called *epitopes*, which protrude from its surface.

The organs of the adaptive immune system called *Lymphoid organs* are positioned throughout the body and lodge the lymphocytes, small white blood cells that are one of the key defenders in the immune system. *Lymphocytes T*, or *T cells*, are one of the many kinds of specialized lymphocytes in the immune system. They mature in the thymus (one Lymphoid organ) and travel throughout the body, using either the blood vessels or their own system of lymphatic vessels. Their surface is covered by randomly generated receptors that can match (recognize) specific antigen's epitopes: each lymphocyte has one kind of receptor which binds to a specific related epitope. In fact, there is a *selection* step of the good T cells that are *immature* at the beginning and learn which proteins of the body are non-self proteins. As most of the self proteins circulate through the thymus, immature T cells that match self proteins in the thymus are destroyed, otherwise, the immune system can wrongly identify self as non-self causing a so-called *auto-immune disease*. This process is called *self-tolerization*. Those T cells that are not destroyed become *mature* and are released in the rest of the body. There, they detect non-selfs and ignore selfs.

3 Stigmergy Paradigm – An Overview

Social insects organize themselves to ensure the survival of the colony by means of a reactive individual behavior and a cooperative collective one. Such behaviors can be observed from different activities: foraging, building nests, sorting larvae, etc. Cooperation in these systems is mediated by an efficient communication mechanism relying on the inscription of task evolution in the environment. This paradigm named ***stigmergy***, introduced for the first time by P. P. Grassé in [6], describes the way social insect communities (ants, termites, bees, etc.) interact through their environment. Schematically, each entity has a local view of its neighborhood but uses a chemical volatile substance (***pheromone***) to mark its environment when achieving a collective task. The pheromone thus deposited is propagated in the environment and *evaporates* with time. The deposit of pheromone creates a *gradient field* in the environment which tends to attract other insects, and to enroll them in a self-catalytic behavior. When the task is finished, no more pheromone is deposited, leading to the disappearance of this information after a period of time through an *evaporation mechanism*. In the particular case of food foraging, each time an ant deposits a pheromone along the path, it reinforces the probability that other ants will choose the same path to reach the food. The amount of deposited pheromone is called the ***pheromonal gradient***, and every ant scanning its neighborhood will have a great probability to walk up the gradient.

4 How the Agents Reproduce the Natural Behaviors

Both ID and IR mimic the behavior of two distinctive natural systems:

- For the detection, the human ***natural immune system***.
  The IDSystem maps some mechanisms borrowed from the immune system. The goal is to couple the efficiency found in the collective behavior of the immune system with the agents' mobility. MAs as T cells, travel around the network and work cooperatively to detect local and distributed suspicious patterns.
- For the response, the social insect ***stigmergy paradigm***.
  More precisely the inspiration is taken from the collective behavior of foraging ants. The present IRSystem maps the collective behavior of a population of foraging ants, using MA as ants and an electronic version of the pheromone.

There are two populations of MAs:

- The ***Intrusion Detection Agents*** (IDAs) incarnate the IDSystem.
  The IDAs borrow mechanisms from immune T cells. To detect attacks happening in the security domain, IDAs responsible for a local domain have to be able to discriminate between normal and abnormal activity. In the immune system it is done by T cells distinguishing selfs from non-selfs. In the IDSystem the normal self activity is viewed as a good sequence of events, whereas the abnormal non-self activity is viewed as a bad sequence of events.
- The ***Intrusion Response Agents*** (IRAs) incarnate the IRSystem.
  IRAs' task is to respond to detected intrusions. For that, they must locate the place where the alarm was given by IDAs and go there. To trace the source of the alert the same mechanism is used as the mechanism employed by the ants to trace the source of food. Ants use the chemical pheromone deposited by the other ants in the environment to trace the source of food. The IRAs use an *electronic version of the pheromone* which indicates the route to the infested node. This pheromone is built by an IDA when it detects an intrusion and is randomly diffused from the infested node through different nodes of the network. Thus, the pheromone allows to build a path that the IRAs follow to trace the source up to the alert. Moreover like in ants' colonies, an evaporation process inhibits the electronic pheromone to limit the number of IRAs following the same pheromonal trails.

Agents' Mobility for Social Organization

- The immune system consists of different cells distributed throughout the body. Each category of cells has a specific function and moves in the body according to its own needs and there is always a co-operation between the cells simulating a "social life of the cells of the immune system". Many cells which play an important part in the immunizing protection move in blood, in lymph, and lymphoid organs to coordinate the various immune reactions. Thus, the concept of a perpetual cellular dynamics is fundamental and without this dynamics and a great mobility the task of protection could not take place.
- In ant populations the pheromone acts as a regulator of the insect traffic outside the nest. Nevertheless, this task of regulation would not be possible without the mobility of the ants. Suppressing the mobility would result in preventing the foraging task leading to the unavoidable death of the ants' colony.

Thus, in each population (immune cells and ants) the mobility is fundamental for the survival of the population. Therefore without mobility it is not possible to reproduce the behavior of these two populations. Intuitively, MAs are good entities to realize the conceptual model.

5 IDSystem

The detection principle is as follows:

The correct execution of different programs and their deviation compared to normal activity are supervised. For that, short sequences of events when the program runs in safe conditions and environment are collected and surveyed, as was done in [2] with short sequences of system calls. For each local security domain, we previously built a *self profile* according to the programs to be supervised. A self profile contains several types of sequences specific to different programs. At the system start-up, when the system runs in real conditions, IDAs created from different hosts have to detect deviation of events specific to these programs. In each domain, different populations of IDAs specific to a program are able to memorize safe sequences obtained from the normal self profile. Each program-specific IDA selects randomly a block of n consecutive sequences and examines in the host they visit, the deviation of the incoming events from the selected set. If the deviation is too high the IDA launches an alert. Otherwise, under a certain level of deviation the incoming sequence is accepted. Moreover, alternatively to normal-abnormal sequence discrimination other populations of IDAs look for specific attack signatures. In this case, IDAs specific to a well-known attack pattern travel randomly in the network to detect this pattern. Here, IDAs refer to signatures previously stored in another *non-self profile* and select one of them. A signature-based IDA that detects an anomalous pattern launches an alert.

Chronologically, an IDA follows the sequence of actions described below:

1. A node houses one self and one non-self profile that have been previously built off-line and stored in two different protected files. The *self profile* contains several types of sequences of events to be accepted and the *non-self profile* contains several types of the sequences of events to be rejected. The self profile is built from normal activity executed in machines previously isolated from the rest of the system. These normal events are called *self events* or *selfs* for short. Each sequence of the non-self profile called *non-self* is attributed a *suspicion value.* This suspicion value indicates the propensity to find such a sequence in the system. In the non-self profile, the different fields are:
   – The *identifier of the suspicious activity*: Id_{sa}. The IDA identifier Id_A is the same identifier as the identifier Id_{sa}, of the suspicious activity it supervises.
   – The *sequence of events*: $\langle(event(1), value(1)), (event(2), value(2)), ..., (event(n), value(n))\rangle$.
   – The *suspicion index*: *SI* that gathers the *degree of importance* of the suspicious activity. *SI* expresses the deviation of the incoming events from the events memorized by the IDA.
   – The *tolerance threshold*: *Th*, which corresponds to the value of the deviation between the incoming events and the events memorized by the IDA, that generates an alarm.

   There are three states in the life of an IDA; it is *immature* at the start-up, then it becomes a *primary* IDA and ends its life as *mature* IDA.

2. At the start-up of the node, first immature IDAs are created by their *generator host*. Each immature IDA memorizes a sequence of the non-self profile and the corresponding fields. It is also attributed a *Time To Live* (TTL) to limit its life duration as immature IDA. The TTL is a positive integer value and at every hop, it decreases by one unit.
3. These immature IDAs are randomly and temporarily dispatched in the system to protect, in order to test their effectiveness: during a short lapse of time expressed by the TTL, any immature IDA that matches exactly an identical sequence of events in the system, is destroyed as bad immature IDA. This step corresponds to a tolerization period and is also present in other immune models such as [7]. In the same way, during the same short lapse of time, any immature IDA that matches exactly selfs encountered in the self profile(s) are eliminated. This corresponds to the so-called *negative selection* phenomenon happening in the body. The negative selection avoids auto-immune reaction in the body. Here, the goal is to anticipate future *false positives.*
4. If it is not destroyed as bad IDA, an immature IDA which TTL reaches zero becomes a *primary* IDA. As primary IDA, its TTL is re-attributed a non-null value and continues its random walk. At each hop it collects entering events and computes the suspicion value. At every hop, its TTL decreases by one unit. If its suspicion value reaches a certain threshold th_m, it becomes a *mature* IDA and it is kept in the system as good IDA. If its TTL reaches zero, it terminates because it is rejected as bad primary IDA. Since it matches no event, it is considered to be unadapted and thus inefficient.
5. As mature IDA, its TTL is deactivated and its last computed suspicion value is maintained. It continues to move with the possibility to adapt its random walk to what happens in its neighborhood. Different strategies are then possible according to the preferences of the security administrator. In the present case, the IDA chooses in its neighborhood the node with the smallest frequency of visit[1] by an identical IDA (identified by Id_A). Each time, it detects an entering event the suspicion value is computed. Beyond the pre-established threshold th_a, an alert is launched.
6. Each time a mature IDA revisits its originating host, the corresponding suspicion value in the non-self profile is updated with the last suspicion value computed by the mature IDA. The originating host is the host that has generated the immature IDA leading to the mature IDA. During the system operation, the number of agents is controlled and they proliferation is limited. This phenomenon is also present in the body where the number of cells and their specialization is controlled to avoid notably a *metastasis phenomenon.* In this case, the cells escape from any control, multiply in an anarchistic way and lose their specificity. However, when it is required cells

[1] Other stategies are also possible.

of the body such as *macrophages*[2] proliferate to protect the body. Similarly, the IDAs' population proliferate only when it is required. Thus, revisiting its originating host, a mature IDA is cloned before moving to the next hop, only if its suspicion value is too high. The two resulting IDAs continue their walk.

7. As soon as a suspicious event is detected the IDA launches an alert, builds an artificial pheromonal information that is specific to the suspicious event which is called *electronic pheromone*. The IDA evades the infested node as soon as possible, freeing the place for other agents especially adapted to perform a response. The IDA, entirely invested in its detection task, changes its walk's strategy; looking for another sign of the same suspicious behavior in its neighborhood, it chooses preferably a neighbor with a pheromone to correlate.

The electronic pheromone comprises all the essential fields dedicated to this sequence of actions. The main fields are:

– The *identifier* of the responsible IDA which built the pheromone after having detected the attack: *IdA*.
– The pheromone creation date: t_0. This pre-suppose that the machines are synchronized between each other.
– The *intermediate date* t_i, which corresponds to the date when the pheromone is deposited on the intermediate node i during the pheromone propagation.
– The *number of hops* that corresponds to the distance in term of nodes at which the pheromone is diffused: *Hop*. The network is represented by a *graph of neighborhood* dedicated to the MAs' displacement. Each MA obtains from the node it is currently visiting the list of the next neighbor nodes it can visit. This neighborhood graph is a *logical graph* built on the top of the real network topology, for the needs of the conceptual model. It represents a logical view of the nodes which can see each other, whereas in reality, in a LAN all the machines can see each other. The neighborhood graph is also used by the SimplePheromonePropagator for the pheromone's diffusion through the network.
– The *gradient* that decreases hop by hop from the alert source and will be used by the IRAs to follow back the pheromonal path to the alert source: *Gd*.
– The *suspicion index*: *SI*, that corresponds to the *degree of importance* of the attack. This degree depends on a *tolerance threshold* fixed at the start-up and which can be adjusted during the system execution.

The electronic pheromone is diffused in a randomly chosen direction by another agents' population, namely the *SimplePheromonePropagators*. Schematically, the SimplePheromonePropagator is called by the IDA before this latter

[2] A macrophage is a type of white blood cell that surrounds the body and kills microorganism, removes dead cells, and stimulates the action of other immune system cells.

evades the infested node. The SimplePheromonePropagator is responsible for selecting randomly a neighbor and moves there to deposit the pheromone. At this new selected node, the SimplePheromonePropagator repeats the same operation: it selects randomly another neighbor it never visited, moves there and deposits the pheromone. The same operation is iterated a number of time corresponding to the field: *Hop*, of the pheromone. This dissociation of roles is quite useful because it allows the different populations of agents to work independently from each other in an asynchronous way. The pheromone is used by the IRAs, another category of MAs living in the system that work cooperatively to respond to the detected attacks.

Overall, the common key points which our model borrows from the immune system are the following:

– The specificity because each IDA is specialized in one kind of anomaly. An IDA memorizes new sequences specific to this anomaly or it is specialized in one specific signature pattern.
– The dynamicity because each IDA continuously circulates through the security domain, which increases the global coverage provided by all the IDAs' population over time.
– The autonomy because each IDA can decide to clone itself only if the level of suspicion becomes too high.
– The distribution of information among the different nodes, which avoids a central point of failure and which allows a distribution of the different points of alert in the system.

6 IRSystem

The response principle is as follows:

At system start-up, another population of agents, the IRAs, is also created at the different nodes. In parallel to the IDAs' population each IRA of the IRSystem performs a random walk through the network looking for a pheromonal trail representing an alert. The roaming IRAs will follow it as soon as they detect its existence. More precisely, this pheromonal information acts as a communication medium to alarm the different populations of IRAs because the IRAs are also specialized in different responses. Therefore, the information carried by the pheromone is different according to the suspected intrusion as well as its seriousness allowing IRAs to perform an adequate response. This is quite advantageous in term of performance because it avoids having inappropriate agents moving to a location where they will be useless. Moreover the pheromone will also evaporate after a certain lapse of time avoiding an IRA to migrate to a location where the alert is obsolete. Finally, as soon as an effective response has been performed a pheromonal inhibition process is activated. This process consists of accelerating the pheromone evaporation process, avoiding that several IRAs follow the same trace.

Chronologically, an IRA follows the sequence of actions described below:

1. At the start-up, it is created by a host and it is in its *normal quiet state.* In this quiet state, it moves randomly and searches for a pheromone in the network.
2. If it finds a pheromone, it switches to a *tracking state* and follows the pheromone to its source.
3. Following the pheromonal trail, it accelerates the pheromone evaporation process on each node belonging to the current trail. That reduces the time remaining before the complete pheromone evaporation without preventing other IRAs from following the same trail. Thus, that increases the chance that a response is given avoiding that too many IRAs follow a trace already detected. As soon as it reaches the node where the suspicious behavior was detected, the IRA performs the response, it destroys the pheromone and it switches to its normal quiet state.
4. Then it continues its random walk to discover another pheromone.

Evaporation of the Electronic Pheromone

Obviously, the evaporation process of the electronic pheromone has to begin in the last node reached by the pheromone. Then it will reach the other nodes of the path in the opposite direction of the diffusion. In this scheme, an IRA visiting a node with a pheromone can always follow the pheromonal gradient up to the first node. An *evaporation index* Δ has been defined as the period of life, allocated to the electronic pheromone deposited in the last node n, before it disappears. This lapse of time is adjusted according to the average time needed by an IRA to perform its task when visiting a node; that is, to read and interpret the pheromonal information. Also to choose the next node to visit. The pheromonal deposit will disappear first at the last node n after a duration Δ.

Then, the pheromone successively disappears at each intermediate node i, starting from the last node n until the node responsible for the intrusion's alert. The pheromone's evaporation date *Tevap(i)* at each intermediate node i has already been computed in a previous paper [5] according to known parameters and is given by:

$$Tevap(i) = t_i + 2 \times \frac{t_i - t_0}{i} \times (n - i) + \Delta$$

where:

- t_i is the pheromone deposit date at the node i.
- t_0 is the pheromone deposit date at the node *0.*

Inhibition of the Electronic Pheromone

To avoid too many IRAs converging to the same source, an *inhibition index*, α, has been defined, that increases the rate of the pheromone evaporation in each node already visited by an IRA. This is explained in the following:

A first IRA entering the node i at $t_{A1(i)}$ should intervene between date t_i and *Tevap(i)*. Thus, after date $t_{A1(i)}$, the time remaining before the pheromone completely evaporates at node i is:

$$Tevap(i) - t_{A1(i)}$$

With the inhibition process, a second IRA should intervene in a period of time corresponding to a ratio α of this remaining time. This inhibited remaining time is given by:

$$\alpha \times (Tevap(i) - t_{A1(i)})$$

Thus, the second IRA has to reach node i at a date $t_{A2(i)}$ such that:

$$t_{A1(i)} \leq t_{A2(i)} \leq t_{A1(i)} + \alpha \times (Tevap(i) - t_{A1(i)})$$

This inhibition process is repeated for every IRA detecting the same pheromone at the same node i, until the pheromone completely disappears or a response is given. Like the evaporation process, the inhibition process contributes to limiting the IRAs' fruitless activity and movement as well as wasteful resource consumption.

7 Conclusion

The conceptual model presented in this paper is built with two main populations of MAs that self-organize to protect a computer network in a distributed and decentralized way. For that, agents have to interact with each other. The goal of this interaction is the emergence of a collective defensive behavior. It makes sense to apply the two natural system metaphors to IDReAM because several characteristics of these two natural systems are present in IDReAM. Notably:

– The natural environment constitutes the communication medium for natural entities; the network environment constitutes the communication medium for MAs.
– Like the natural entities (T cells, ants), agents are immersed in the environment.
– Thanks to the environment, inter-entity interaction leads to a collective communication model that deals with large-scale tasks such as the detection of a source of food for ant populations or the detection of a source of attack for MA populations.
– Both types of systems (natural systems and MA systems) behave dynamically and each state influences the environment. In its turn, the environment influences the system dynamics. Thus, the global system equilibrium is maintained by an interaction loop between the environment and its population.

As it was previously mentioned, the present paper details mainly the conceptual model of IDReAM. However, IDReAM's assessment is provided in a PhD Thesis [3] with respect to the following requirements:

1. Robustness: The robustness is mainly due to the furtivity of the agents distributed in the network. There is no point of centralization of all the information because, according to the conceptual model the majority of the information is located at different locations of the network environment; each machine owning its set of profiles (sel, non-self), each pheromonal trace being deposited temporarily and in a random way in the network. Moreover, in case of accidental death of the agents (IDAs and IRAs) a mechanism that controls the population of agent enhances the model; this mechanism ensures that the visits' frequencies of the agents (IDAs and IRAs) at each node do not exceed a certain value (or is not under a certain value): if agents are missing the hosts create new agents, if there are too many agents the hosts destroy them. Moreover, all the agents are encrypted when traveling through the network and they are authenticated by the hosting node before being executed. This prevents them from being corrupted by an attacker.
2. Extensibility: IDReAM can be easily extensible to detect new intrusions by adding new sequences in the non-self profiles without disturbing too much its operation. If the intrusion is extremely suspect it suffices to attribute a high suspicion value to the corresponding non-self sequence.
3. Scalability: The model ensures the scalability of IDReAM for two reasons: Even if there is a great number of immature IDAs just a few of them succeed as mature IDAs. Moreover the number's effect is reduced because the moving agents (IDAs and IRAs) are permanently distributed among the machines. Moreover, the model guarantees that small MAs providing simple and lightweight detection and response tasks do not undoubtedly lead to the degradation of the performances.

In the same way, [3] provides the description of IDReAM's implementation using a pure Java-based mobile platform named J-Seal2 [8]. The behavior of the different populations of agents and the corresponding services in accordance with the model have been experimented. The results obtained are rather promising and convinced us of the feasibility of the conceptual model to build IDReAM and the reader is invited to refer to [3] for the implementation and the assessment details. Certain results were already presented in [4]. A recent benchmark has been provided that will make the object of a forthcoming article.

References

1. G. Di Caro and M. Dorigo. Ant colonies for adaptive routing in packet-switched communications networks. *IRIDIA Universit Libre de Bruxelles Belgium*, 1998.
2. S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff. A sense of self for unix processes. *In Proceedinges of the 1996 IEEE Symposium on Research in Securit and Privacy Context Related*, 1996.
3. N. Foukia. *IDReAM: Intrusion Detection Executed with Agent Mobility - A Distributed Approach Inspired from Natural Life Systems.* PhD thesis, University of Geneva, 2004.

4. N. Foukia and S. Hassas. Managing computer networks security through self-organization - a complex system perspective. In Proceedings of the First International Workshop on Engineering Self-Organising Applications (ESOA), Melbourne, Australia, 14-15 July 2003.
5. N. Foukia, S. Hassas, S. Fenet, and J. Hulaas. An intrusion response scheme: Tracking the alert source using stigmergy paradigm. *SEMAS 2002, Bologna - Italy*, July 2002.
6. P. P. Grassé. La reconstruction du nid et les interactions inter-individuelles chez les bellicoitermes natalenis et cubitermes, la thorie de la stigmergie - essai d'interprtation des termites constructeurs. *Insectes Sociaux, no. 6*, pages 41–81, 1959.
7. S. A. Hofmeyr and S. Forrest. Immunity by design: An artificial immune system. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Morgan-Kaufmann, San Francisco, CA*, 1999.
8. J-Seal2. http://www.coco.co.at/development/.
9. S. H. Kleinstein and P. E. Seiden. Simulating the immune system. *Computing in Science and Engineering*, pages 69–77, July 2000.
10. A. Martinoli. *Swarm Intelligence in Autonomous Collective Robotics: From Tools to the Analysis and Synthesis of Distributed Collective Strategies.* PhD thesis, EPFL, Lausanne, Switzerland, 1999.

Managing Dynamic Flows in Production Chains Through Self-organization

Frederic Armetta[1,2], Salima Hassas[1], Simone Pimont[1], and Emanuel Gonon[2]

[1] LIRIS, Nautibus, 8 Bd Niels Bohr,
Université Claude Bernard-Lyon 1,
43 Bd du 11 Novembre F-69622 Villeurbanne
[2] OSLO-Lyon, 40 rue Camille ROY,
69007 LYON

Abstract. In this work, we are interested in modeling a production chain following the perspective of complex adaptive system. We propose an approach, allowing a production chain to manage by itself, its own behavior, so as to satisfy the constraints imposed upon it by its environment and reach a set of predefined objectives. We propose self-organization as a mechanism, to achieve such a goal.

Keywords: self-organization, emergent behavior, swarm intelligence, software engineering.

1 Introduction

In many industrial manufacturing systems, the great number of tasks to achieve for each product, the great number of products to produce, routing flows of products in the system, etc , are at the essence of the system complexity. In addition, the system evolution depends on the variation of many factors such as : the amount and kind of products, products priorities, the state of resources (machines breakdowns, operators delays, etc). In this work, we are interested in modeling a production chain following the perspective of complex adaptive systems. We propose an approach, allowing a production chain to manage by itself its own behavior, satisfying the constraints imposed upon it by its environment and attempting to reach a set of predefined objectives. We propose self-organization as a mechanism, to achieve such a goal.

1.1 Context and Objectives of Our Case Study

To manage the manufacturing, we focus on three main elements :

– Tasks : A product manufacturing consists of achieving a sequence of tasks.To make a product, one must achieve all its tasks.
– resources : represented by machines which achieve the tasks necessary for a product manufacturing. They have a limited capacity and can break down.

– Flows : A flow is associated with a product type. It is characterized by a progressing speed (how long does it take to achieve a production stage) and a size (how many products are realized at the same time) for each of the production stages. Its evolution depends on all the product dispatching decisions made by the production management system.

We consider a production chain in the domain of electronics. For this domain, we can notice the following characteristics :

– a long sequence of operations (more than a hundred) for a manufacturing product;
– reenterant flows: which means a manufacturing product uses several times the same resource for its processing;
– the use of evolving technologies;
– the use of expensive resources;
– the use of same resources for both manufacturing products and research and development products;
– frequent breakdowns;
– evolving commercial needs;

In this context, the production chain needs to satisfy the following (sometimes opposite) objectives :

– Objective 1 : maximize the resources use, and maintain this maximization on a long term;
– Objective 2 : make the output of the production system linear ;
– Objective 3 : minimize the mean time of the production cycle (the production cycle corresponds to the duration of a product manufacturing) ;
– Objective 4 : make the system attribute a high priority for the manufacturing of research and development products;
– Objective 5 : and at a longer-term, acquire information, from the system itself, on its own feeding requirement. This allows to provide a kind of system load profile (composed by new products to make).

1.2 Guidelines and Suggestions

To reach the objectives stated above, we propose the following guidelines, that will serve as founding concepts for our approach.

– To maximize the resources use (Objective 1), we propose to ensure a sufficient amount of products available on each resource (queues of products ahead of resources). This makes the system robust and allows for the absorption of perturbations that may occur in the system. Even if a resource is not anymore feeded with products, it can manufacture products in its queue. To do so, we suggest to make the system behave by anticipation, forecasting its future needs. The system do so, by acquiring continuously, qualitative information on its activity.

- to make the system's output linear (Objective 2), we suggest to control the progression of the products in the system. To do so, we use a mechanism of information feedback, based on the requirements in term of system activity and needs, to obtain an ideal system output. The system attempt to satisfy a linear output.
- to minimize the mean time of the production cycle (Objective 3), we suggest to maintain a small number of products in progress in the system. To do so, we seek to schedule tasks on adequate resources (in a predefined objective time window), so as to obtain a satisfactory mean number of products in progress [1] .
- to give a priority to the manufacturing of research and development products (Objective 4), we suggest to make the system be able to control the progress of products to be manufactured;

To address these questions, we propose to model the production chain as a complex adaptive system, which activity is founded on the guidelines described above. Our complex adaptive system is represented by a situated multi-agents system. Thus, the production system is represented as a situated multi-agents system, the objective of which is to find a spatial organization making it possible to satisfy the environmental constraints, while addressing the flow management problem. This paper is organized as follows : first we give our motivation for the use of a situated multi-agents system, we present various approaches that have been applied to the production management. In section 3, we present our approach and underlines its founding concepts. In section 4 we present some experiments and results. We conclude this paper by a discussion and by giving some perspectives for the ongoing work.

2 Related Works

There is a wide range of works applied to the scheduling management. For this problem, we distinguish two solving philosophies. For the first one presented in 2.1, we present approaches based on centralized systems to efficiently compute a schedule to apply. In 2.2, we present approaches which performances come from empirically correct behavior.

2.1 Optimization Methods

Optimization methods have frequently been applied to the typical job-shop problem. Different approaches exist such as "Branch and Bound" methods and the constraint satisfaction methods [3][13], methods based on mathematical analysis [9], use of metaheuristics for efficiently covering the search space [1][22], their

[1] Average waiting time of products between two production stages (induced by the predefined objective time window) is related with the amount of products in the system. Large waiting time = several products in queues = large amount of products in progress in the system.

combination to some classical optimization approaches [19], etc. In [14], an hybrid approach combines various method types for optimization and produces good results.

Although giving an optimal schedule that minimizes (or maximazises) a performance measure f (usually a function of job completion times), the above presented methods lack robustness, when faced to disturbance, and don't fit the industrial problem. Computing this kind of optimal schedules is useless if they can't be applied because of many real-time disturbances. New methods are developed in order to accommodate disturbances. The group 'Robustesse et Flexibilité' works on generating robust scheduling with uncertainties using methods from operational search and artificial intelligence [5]. In [8], authors looks for super solutions allowing the integration of local disturbances. These integration attempts are difficult to realize and can not be applied at the present time to all scheduling management problems.

2.2 Empirically Correct Behavior Methods

The scheduling management can be the result of choices realized in real time by the system. Thus, empirically correct behavior methods define the waiting queue product priorities and the route realized by each product, at each product's stage manufacturing. By its intrinsic properties (several kinds of perturbations, several interactions, etc), a production chain exhibits the characteristics of a complex system, evolving in an uncertain environment. The multi-agents paradigm (MAS) provides good characteristics (such as adaptation capacities, decentralized control, autonomy, etc), for representing such a system. In MAS, the contract-net protocol is a negotiation mechanism that allows products to compete for resources [21]. Cooperation between products competing for acquiring resources increase system performances [20]. Parunak and Barto discuss in 'Agent-based Models and Manufacturing Processes' the significant improvement realized by agent acting for the electronic components manufacturing in an AMD factory. In [2], the resource specialization for tasks allows to increase system performances. In [18], authors use a more cognitive approach called Methamorph. In this case, negotiation is made between hierarchically arranged agents from different types having important deliberating capacities. In [17], authors consider the learning capacity of such systems.

In order to avoid bottleneck, one must control Work-In-Progress (WIP) level in the production system [11]. In [4], authors try to maintain an appropriate WIP level on each resource. Thus, the priority products in queues are defined according to products usefulness for the system. In such a way, in [15], authors control upstream and downstream WIP level on each resource. These methods allow to communicate reception capacity towards upstream production layers, to maintain good performances for the system.

In [16], authors propose to provide system bottlenecks to a centralized algorithm which dispatch products in order to decrease congestions impact. Putting up the number of known bottlenecks integrated to compute the dispatching increase the process efficiently.

Although the control of the WIP level allows a limited control of the system trend, presented methods do not anticipate flows products through the system (MRP2 logic in industrial area). However, we think that we can take advantage of a multi-agents approach realized at a bottom layer (representing each planning task to realize by an agent). These modeling allows to anticipate congestions and to take advantage of an implicitly decentralized representation of the system state. Then, it is not necessary to indicate to the system the bottlenecked resources because they are known by the system itself. [10], presents a fuzzy anticipating planning which evolves each time a disturbance occurs. This forecasting technique provides useful information to the decision making process.

3 Our Approach: A Self-organizing Approach for Dynamic Flow Management in a Production Chain

In our approach we address the problem of production scheduling, as a spatial negotiation process between a set of tasks and a set of free scheduling temporal areas on the available resources. To do so, we use a metaphor of fighting between competing reactive agents. Our approach is based on the following principles :

3.1 Eco-Resolution

The eco-resolution paradigm has been first introduced by J. Ferber in 1989 [6]. It bases the resolution of a problem on mechanism of a state space search, by reactive agents guided by a satisfaction fitness function. In our problem, we represent the product tasks as situated agents, evolving in a shared environment corresponding to the set of available scheduling temporal areas on the system resources. In order to build the schedule, agents have a unique goal, which is to place themselves as soon as possible on suitable scheduling temporal areas defined on their environment [2]. The environment corresponds in fact to a zone of anticipation of the scheduling, that is gradually realized by the resources in the course of time. Although environment corresponds to a temporal planning (each task agent is placed with a begin/end date), we describe it as a spatial environment.

3.2 Stigmergic Communications and Application

Stigmergy is an indirect mode of communication, mediated by the environment. This principle has been first introduced by Grassé [7], while studying the behavior of social insects. It provides a spatial distributed coordination mechanism. In our approach, task agents, are competing agents, which try to find satisfying places on their environment. The agents placed form different local patterns with various topologies. The space characteristics of the patterns thus formed influence the behavior of the agents seeking to improve their situation. Thus, there is a phenomenon of positive/negative feedback between the patterns structures

[2] Agents place themselves earliest in the planning of a resource.

evolution, resulting from past behaviors of agents, and their future behavior. Thus, let us see a way of putting in competition agents by stigmergic way.

3.3 Self-organization Through a Fighting Mechanism

To find a satisfying place in the environment, a task agent visits randomly the set of suitable resources.[3] On a selected resource, the task agent tries to get a scheduling area, by competing with tasks already scheduled on the resource. This competition between agents is aimed to maximize the resources use. Unused scheduling areas are thus pushed towards a later scheduling time.

We can see on figure 1, how a spatial negotiation is achieved by agents, at the level of a resource. On the left part of the figure, we represent the resource structure, which is composed of a set of scheduled task agents and a set of unused scheduling areas. A task agent, seeking for a suitable scheduling area creates different groups composed of sets of placed agents and unused scheduling areas, fitting its needs. Then, the attacking agent randomly selects a group with which it could engage in a fighting process.

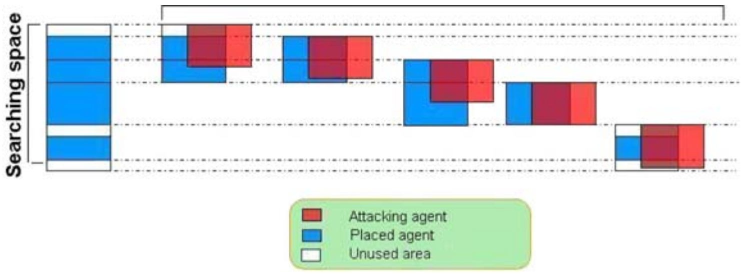


Fig. 1. Fighting mechanism of tasks agents

If the first component of the selected group is an unused scheduling area, the attacking agent wins the fighting. This mechanism of fighting allows for efficiently pushing away unused scheduling areas present on a resource. As the number of fighting achieved by competing tasks agents is high, the unused scheduling areas weakening the system's environment are pushed out of the planning. This phenomenon tends to maximize the system resources use.

3.4 Considering Information Embodied in the System

A Need to Refine the Generated Solution. The mechanism presented above responds to the first objective of our problematic (i.e. flow managing to maximize resources use). However, to respond to the third objective (reducing the amount of products in progress in the system), one has to reduce the mean

[3] A resource is suitable for a task agent, if it is able to achieve the corresponding task.

length of waiting queues of products. This reduction increases the risk to under-utilize a resource, for lack of products [4]. Thus, we must make a good balancing between satisfying objective 1 and objective 2. To guarantee that the products arrive on resources at the appropriate time (avoid congestion), one has to anticipate the flows produced in the system in order to find a solution allowing as well as possible to satisfy the resources capacities on the long term.

What Kind of Decision has to be Made? When the size of a resource waiting queue becomes important, a congestion happens, making the resource capacity insufficient to feed other resources located downstream. One has thus, to carry out choices of priority between tasks, at the level of the congested resource. These choices, allow some tasks to place themselves on the resource, so as to manufacture urgent products quickly, and let less urgent tasks in the resource waiting queue.

Listening to the System Activity. In order to define efficiently the priorities of agents in queues, we must consider information embodied in the system. Indeed, it is possible to consider the capacities of the resources on an extended horizon of production, and be able to feed the system resources with adequate products at adequate times. Placements carried out by anticipation in the so described ecosystem, provide us information on needs to fill. Also let us notice that the mechanism that we will describe could also make it possible to linearize the products output of the system (objective 2). It is then sufficient to consider the linear output of the system as a regular capacity to fill, with the finished products outgoing of the chain of production.

Two Kinds of Information. In order to refine the placement mechanism during the fighting process between competing agents, we propose to consider information revealing the importance of the different competing agents. On one hand, already placed agents will be able to benefit from supporting information, which reveal to the system, the importance of their current placement. On another hand, agents seeking a potential placement, could benefit from information on their urgency for the production process, and then on the necessity they have to be scheduled quickly. This mechanism of recruitment of the useful products supports the insertion of the not yet placed products.

These two kinds of information, goes up the planning gradually, to inform agents and guide the different fighting process taking place in the system. Thus, one collects within a resource, qualitative as well as quantitative information.

Informations About the Utility of Placed Agents. In order to allow already placed products to maintain their position, we make these agents profit

[4] High product in progress level allow to flood the schedule (important queues on resources). Scheduling with low product in progress level is more difficult and less robust face to disturbances like resource breakdowns.

from a kind of information corresponding to their usefullness for the system. This information is generated by each task agent, placed in the environment. To maintain its placement, an agent must also allow to maintain the placement of agents corresponding to the preceding tasks of the product which it represents (A task agent can be placed only if the completion date of manufacturing of the preceding stage exists). Thus, with this first kind of information, placed agents inform precedent agents of theirs associated products, of their usefullness for the system.

Information About the Need of the System to be Filled by not yet Placed Agents. This mechanism allows resources lacking of products, to inform the corresponding tasks, of their urgency for the manufacturing process, and boost their activity. Information is generated from an unused temporal area (free area) on the resource schedule. On this resource, some products with various stages of their manufacturing process, are produced. Information is first of all characterized by a type of product with one of its manufacture stages on the resource. It is this combination *product-stage* which defines the course that will be carried out by this information in the system. Gradually, information traverses backward way, stages of products of the same kind as the required product (a stage is associated with a resource). At each stage of the course, if an appropriate task is found, the traversing information is transmitted to it, and the course of traversing information is over. Let us notice, that the traversing information could move until the stage 0 of the product manufacture, and thus could represent a desirable loading profile for the system. This information could thus also be considered to make decisions, to insert new products in the system (objective 5).

The importance of transmitted information is computed locally. This information importance could, for instance, be based on the size of the unused area on the resource. Indeed, the more an unused area is large, the more it seems important to attract products on the site of free areas in order to dissolve the congestions being formed in the system.

As the system evolves, generated information may also evolve. To maintain collected information significant during the system evolution, we have introduced a mechanism of evaporation, which makes non reinforced information disappear.

Considering New Information Type During the Fighting Process. The fighting mechanism has to be refined in order to use effectively this sum of information, generated by the system, while allowing its stabilization. The basic fighting mechanism is refined, in order to consider this kind of information. Thus, the attacking agent carrying out the attack, wins the fighting inevitably if its 'supposed usefulness' for the system, is higher than the highest usefulness of the placed agents, to which a threshold of movement is added. This allows to reduce the system activity. In case the supposed usefulness of the agent is lower, one applies then the basic fighting mechanism. When two agents belonging to the same type of product are engaged in a fighting process, the attacking agent never wins. Indeed, when applying a policy of management of flows , two agents belonging to the same flow of product do not find it beneficial to fight each other.

4 Experiments and Results

In this section we present some experiments and results obtained through simulations. We have led first, some experiments to test the application of the simple fighting mechanism, presented in section 4.1. In 4.2, we could see how the use of traversing stigmergic information through the system provides interesting improvements for the system activity. We have focused our experiments only using one kind of information described in 3.4 : information related to the importance for the system, of the different placed tasks agents.

4.1 The Basic Fighting Mechanism

We describe here, two characteristics that we have observed, just using the basic fighting mechanism (without using additional information). The first characteristic concerns the self-organization of product flows through the system resources, so as to avoid congestions. The second one concerns the ejection of unused scheduling areas, to a more delayed scheduling time.

Self-organization of Dynamic Flows of Products

- Experiment
  Using the basic fighting mechanism, we try to make the flows of products in the system to take place, and organize themselves , so as to avoid resources congestion. Experiments have been led with products, that have various choices of resources at some stages of their manufacturing. Thus, we can see on figure 2 , that product $P1$ can achieve its different stages of manufacturing on Resource $M0$ or $M5$. On the other hand, the product $P0$ can carry out some of its stages of manufacturing only on $M0$.
- Results
  We can see on the graph, when there are only products of $P1$ type in the system, task agents of different $P1$ products, are placed equitably on $M0$ and $M5$. Indeed, each time an agent seeks a place, it selects randomly a resource that can satisfy it. This implies that on average we obtain half of the tasks placed on $M0$ and the other half on $M5$. While inserting $P0$ products in the system, we could notice that task agents, associated with the different $P1$ products, have tendency to fix themselves more easily on the least attacked resource (i.e. on $M0$). Indeed, on the most attacked resource, the environment is often evolving because of the many undergone attacks, and adherence is less important than on resources less prone to attacks.

  This mechanism allows $M0$ to be naturally discharged from $P1$ products. It acts as a natural mechanism of production load balancing. Thanks to the variable horizon of the scheduling space, this mechanism of balancing is carried out by anticipation. This makes the system stabilize itself towards a solution that reduce resources congestions.

Ejection of Unused Scheduling Areas. The various experiments carried out enabled us to notice that the basic fighting mechanism makes it possible

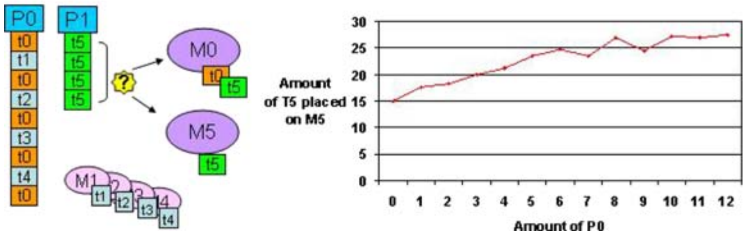


Fig. 2. Natural protection of the bottlenecks

to push back effectively unused scheduling areas present on schedules of the various resources. Indeed, we support the ejection of these unused areas , thanks to the rule defining that an attacking agent (wishing to obtain a site), wins the fighting against a group of preys, if the first element of the attacked group is an unused scheduling area. The activity of the system evolves naturally to a solution which minimizes the amount of unused scheduling areas on the various resources. Indeed, the system is stabilized when no not yet placed agent can be integrated into a schedule because no unused scheduling area allows insertion. Also, faithful to the principle of eco-resolution, none of the already placed agents can improve its position because of lack of unused scheduling areas, in the system.

Rebuilding the Solution Face to Perturbations. In addition of the two above characteristics, we were interested in the way in which the system supports the disturbances occurring in real time. Thus, we simulated various breakdowns of resources. We took for principle that when a failure occurs on a resource, all the agents placed on this resource are ejected. Thus, we noticed that according to the principle of eco-resolution, the various agents seek again a new satisfying placement. This mechanism of search is carried out while again pushing back effectively the unused scheduling areas, until stabilizing itself towards a satisfying solution.

4.2 The Refined Fighting Mechanism

In this part, we present various simulations allowing us to study the interest of the integration of additional information, related to the state of the system, in the course of the computation. We will thus see, how and why integrating this information guides the system to a satisfying solution. Various simulations which we could carry out enabled us to notice that for each one of types of fighting used, the system reach the stabilization. However, by using additional information, the setting time of the system is more important. The system indeed is stabilized, only when the situation obtained supply a place corresponding to its supposed usefulness for the system to each task agent. We did not met, for the moment, a cyclic situation preventing the stabilization of the system. We do not exclude this possibility in some cases. Also, we think of deepening the study of the system activity as suggested in [12].

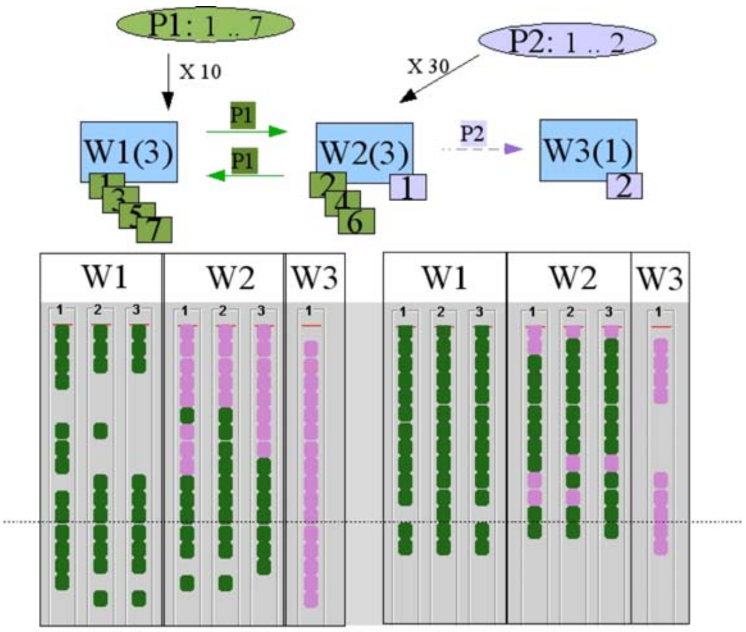
Context of the Experiments. In the presented results below, we focus on maximizing the production system use. We think, however, that the described processes could be integrated in a policy which aims to linearize the production system output as well as to minimize the products in progress level inside the system. For now, we have considered the integration of one kind of information, concerning usefulness for the system of placed agents. To do so, for each experimented situation, we have computed some indexes revealing the importance of the improvement achieved with this information. Theses indexes are not aimed to give a performance evaluation of the three studied problems. For our experiments, the workshops have variable capacities of production relating to the number of machines which they comprise. The products P1 have to be done in a reentrant manner, while the products P2 can saturate the machines.

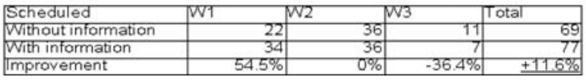
Maximizing the Production System Use

– Experiment
  For this first simulation, we wished to study the distribution of two flows of products in the system depending on its capacity (Figure 3). Thus, flows of products $P1$ and $P2$ traverse various workshops which could carry out their various operations. Workshop 3 has a capacity of 1, whereas other system's workshops have a capacity of 3. The flow of products corresponding to $P2$ thus knows an output of 3 for its first stage and only of 1 for its second stage. It appears thus logical that the flow of product 1 ($P1$ flow) moves more quickly in the system than flow of product 2. In real situations, this kind of bottleneck appears in an unforeseeable way in the system because of various disturbances. It is then advisable to reorganize flows in order to maintain good performances for the system.

  > With approaches based on 'contract net' protocol, one can setup rules allowing to manufacture in priority, on workshop 2, products not being blocked with their following stage. We wish to make our system subject to the same kind of problem as those solved by the use of 'contract net' protocol, by using however, a vision by anticipation.

– Results
  On the left part of figure 3, we can see the scheduling achieved by our system, using the basic fighting mechanism. We could note that some $P1$ products do not manage to be planned on the level of their stage 2 whereas they can be scheduled on workshop 1 for their stage 3. Conversely, some $P2$ products are scheduled on workshop 2 for their stage 1 whereas they are blocked with their stage 2 because the system cannot accommodate them. These $P2$ products appear thus less urgent but opportunism resulting from the basic fighting mechanism allows them to maintain their position. We see on the right part of the figure that the integration of additional informations makes it possible to solve this defect. Indeed, while using this information, all the $P1$ products manage to be scheduled. Transmission of information concerning the placement of agents makes it possible to know which agents





Scheduled	W1	W2	W3	Total
Without information	22	36	11	69
With information	34	36	7	77
Improvement	54.5%	0%	-36.4%	+11.6%

Fig. 3. Distribution [without/with] information : Distance of level 1

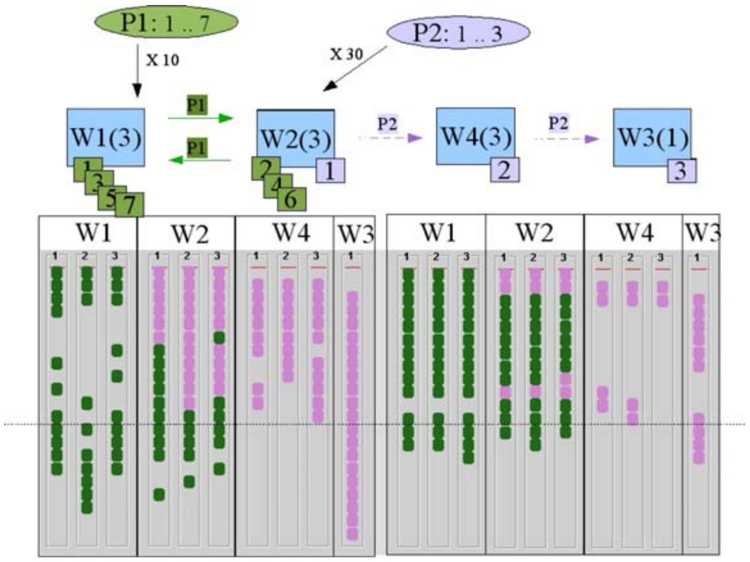
are useful for the system. Placed agents, with low usefulness for the system, loose their positions. Consequently, the overall performance of the production system is maximized. One notices, on another hand, that each of $P2$ products scheduled on workshop 2 are also scheduled with their following stage. These placed agents are regarded as useful for the system and manage to preserve their positions.

Flows Crossing on a Wider Horizon

– Experiment
  We wished to measure the performance of our system over the same type of problem of flows crossing, but this time on a wider horizon (Figure 4 [5]). Thus, flow 2 corresponding to the $P2$ product undergoes a bottleneck on its third stage level. The principle of distribution must remain the same one as previously. It is useless to place products on workshop 2 if these products cannot be accommodated by the system for their following stages. Thus, without any modification of the behaviors used, we subject our system to this new problem.

[5] The various tasks placed on the figure 4 do not take part in the computation of the various indexes because, taking them into account does not reveal the coherence of the placement obtained.

– Results
  Information forwarding in the system, once more, allows the system to reach stabilization on a good quality solution. The system is not disturbed by this more distant horizon because the system carries out a schedule by anticipation which enables him to already anticipate the problem on the level of workshop 2. Moreover, information evolving in the system does not have a horizon of limited diffusion. On the contrary, protocols such as 'contract net' have more limited vision which does not make it possible to anticipate easily and dynamically this kind of problems. Thus, on the left part of the figure, we can see that by using the basic fighting mechanism, one obtains the same kind of result as presented previously. In the same way, on the right part of the figure, we notice that the system adapts well to the new problem: only the awaited products can be placed.

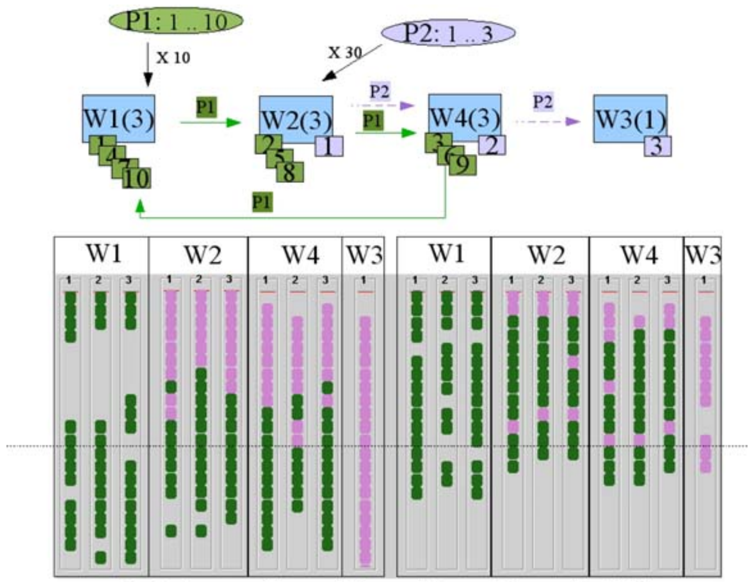


Scheduled	W1	W2	W4	W3	Total(-W4)
Without information	17	36	26	10	63
With information	33	36	10	7	76
Improvement	94.1%	0%		-30%	20.6%

Fig. 4. Distribution [without/with] information : Distance of level 2

Exploring the Solution Space Capacity

– Experiment
  At last, we wished to increase the difficulty of the problem in order to have a better idea of the capacity of the system, in exploring the solution space (Figure 5). Indeed, for this simulation, the $P1$ products carries out also some of its tasks on workshop 4. Thus, the two flows ($P1$ and $P2$) are competing for the workshops 2 and 4.



Scheduled	W1	W2	W4	W3	Total(-W4)
Without information	17	36	32	10	63
With information	31	36	32	8	75
Improvement	82.4%	0%		-20%	19%

Fig. 5. Distribution [with/without] information : Distance of level 2 + difficulties

– Result
  On the left part on figure 5, we see the result obtained by using the basic fighting mechanism. By opportunism and because of a new range for $P1$ products allowing it, all the tasks of the 10 $P1$ products inserted in the system manage to be placed on workshop 1 (10 * 4 stages = 40). However, these placements are not optimum because on workshop 1, many scheduling area remain unused. $P1$ products are placed too late on workshop 2. Places being once more occupied by $P2$ products (blocked with their stage 3), and thus being less urgent. On the right part of the figure, we see that by using the refined fighting mechanism, the solution is again of good quality. The activity of the system allows the agents of $P1$ to have a chance to be placed on the workshop 4 which is a congested workshop. These products were able to exploit their chances and thus to stabilize their positions and to improve it, in order to get a place according to their importance for the system. One can thus say, that at the system level, there is a feedback between the attempt of placement and its validation by the system. All this, generating in addition, an activity in the system, allowing the $P1$ agents to have an opportunity to be placed on workshop 2 and 4, so that they can then also stabilize their situation and improve it. The system is stabilized finally when each product obtains a position which is suitable for him.

Need for More Boosting Information. In the various tests carried out, the natural activity of the system allowed each agent to find a suitable place.

However, we think that in some cases, for lack of activity, information will have to be brought to the products blocked (not-scheduled) on the different workshops in order to allow them to have a chance and to prove their importance for the system. This second type of boosting information corresponds in fact to information relative to various unused scheduling areas, present in the previously described system (3.4). In addition, because of the mechanism of information used, the task agents corresponding to last tasks of their product, have a weak level of information because they do have no (or not enough) agents located downstream bringing weight to them. In order to return to the task agents, situated at an end of manufacturing process, all their importance, one will be able to use a mechanism allowing, to inform these tasks that they correspond to awaited finished products, and this following a regularity of output of finished products for the system. This last point, allows to respond to the objective of making linear the system outputs (Objective Two).

5 Conclusion

We have presented in this paper, an ongoing work on the dynamic flow management in a production chain, through mechanisms of stigmergy and self-organization. Our approach is based on the use of the situated multi-agents paradigm, reproducing a fighting mechanism between competitive agents. We have first proposed a basic fighting mechanism to achieve dynamic scheduling of tasks on a set of resources, in order to avoid congestion of the system resources and make flow of products more flexible. We have also observed through simulations, robustness of the system face to perturbations. We have proposed at a second step, a refinement of the basic fighting mechanism, through the integration of retroactive information inside the system. This last mechanism permits to counterbalance the opportunist character of the first one. We have presented some experiments and results. The first obtained results seem interesting. We are now deepening our study in order to formalize our approach.

References

1. M. Ventresca B. Ombuki. Local search genetic algorithms for the job shop scheduling problem, November 2002.
2. Bonabeau E Theraulaz G & Deneubourg Campos, M. Dynamic scheduling and division of labor in social insects. *Adaptive Behaviour*, 8:83–95, 2001.
3. Yves Caseau and Francois Laburthe. Improving branch and bound for jobshop scheduling with constraint propagation. In *Combinatorics and Computer Science*, pages 129–149, 1995.
4. Yun-Min Feng Chih-Hung Tsai and Rong-Kwei Li. A hybrid dispatching rules in wafer fabrication factories. *International journal of the computer, the internet and management*, January 2003.
5. E. Sanlaville et Al. Flexibilité et robustesse en ordonnancement, article du groupe flexibilité. *ROADEF*, 2002.

6. Jacques Ferber. *Objet et Agents : une étude des structures de représentation et de communication en Intelligence Artificielle.* Thèse de doctorat, Université Paris VI (Jussieu), 1989. In French.
7. P.-P. Grassé. La reconstruction du nid et les coordinations inter-individuelles chez *Bellicositermes natalensis* et *Cubitermes sp.* La théorie de la Stigmergie: Essai d'interprétation du comportement des Termites Constructeurs. *Insectes sociaux*, 6:41–80, 1959. In French.
8. E. Hebrard, B. Hnich, and T. Walsh. Super solutions in constraint programming. In *Proceedings CPAIOR'04*, Nice, France, 2004.
9. X.. Zhao J. Wang, P. B. Luh and J. Wang. *An Optimization-based Algorithm for Job Shop Scheduling*, volume 22, chapter 2, pages 241–256. 1997.
10. H. Van Dyke Parunak John A. Sauter. Ants in the supply chain, 1999.
11. C.G. Panayiotou and C.G. Cassandras. Optimization of kanban-based manufacturing systems. *Automatica*, 35:1521–1533, 1999.
12. H. Van Dyke Parunak, Sven A. Brueckner, Robert Matthews, and John Sauter. How to calm hyperactive agents. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 1092–1093. ACM Press, 2003.
13. Franois Roubellat & Al. Pierre Lopez. *Ordonnancement de la production.* Hermes, 2001.
14. Dardilhac D. et Dezalay D. Portmann M-C., Vignier A. Branch and bound crossed with ga to solve hybrid flowshop. *European Journal of Opérational Research*, 107:389–400, 1998.
15. K. Schild S. Bussmann. An agent-based approach to the control of flexible production systems. In *Proc. of the 8th IEEE Int. Conf. on Emergent Technologies and Factory Automation (ETFA 2001)*, 2001.
16. Subhash C. Sarin and Sameer T. Shikalgar. Reduction of average cycle time at a wafer fabrication facility, 2001.
17. Weiming Shen, Francisco Maturana, and Douglas H. Norrie. Learning in agent-based manufacturing systems. In *in Proceedings of AI & Manufacturing Research Planning Workshop*, pages 177–183. The AAAI Press, 1998.
18. Weiming Shen and Douglas H. Norrie. Combining mediation and bidding mechanisms for agent-based manufacturing scheduling. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 469–470, New York, 9–13, 1998. ACM Press.
19. V. T'Kindt, N. Monmarche, D. Laugt, and F. Tercinet. Combining ants colony optimization and simulated annealing to solve a 2-machine flowshop bicriteria scheduling problem. In *13th European Chapter on Combinatorial Optimization (ECCO XIII)*, pages 129–130, Mai 2000.
20. John M. Usher and Yi-Chi Wang. Negotiation between intelligent agents for manufacturing control, 2000.
21. Yi-Chi Wang and John M.Usher. An agent-based approach for flexible routing in dynamic job shop scheduling. In *11th Industrial Engineering Research*, 2002.
22. Xiaokun Zhang Yuefei Xu, Robert W. Brennan and Douglas H. Norrie. A genetic algorithm-based approach to holon virtual clustering. In *in Proceedings of World Multiconference on Systemics, Cybernetics and Informatics (SCI'2000)*, pages 380–385, 2000.

A Self-organizing and Fault-Tolerant Wired Peer-to-Peer Sensor Network for Textile Applications

Christl Lauterbach[1], Rupert Glaser[1], Domnic Savio[1], Markus Schnell[1], Werner Weber[1], Susanne Kornely[2], and Annelie Stöhr[2]

[1] Infineon Technologies AG, Corporate Research, Otto-Hahn-Ring 6
81739 Munich, Germany
christl.lauterbach@infineon.com
[2] Siemens AG, Corporate Technology, Otto-Hahn-Ring 6
81739 Munich, Germany

Abstract. Textiles are omnipresent in everyday life. Their combination with microelectronics will lead to completely new applications, thus achieving elements of ambient intelligence. The integration of sensor or actuator networks, using fabrics with conductive fibres as a textile motherboard enable the fabrication of large active areas. In this paper we propose a "smart textile" based on a wired peer-to-peer network of simple information processing elements with integrated sensors or actuators. A self-organizing and fault-tolerant architecture is accomplished which detects the physical shape of the network. Routing paths are formed for data transmission, automatically circumventing defective or missing areas. The network architecture allows the smart textiles to be produced by reel-to-reel processes, cut into arbitrary shapes subsequently and implemented in systems at low installation costs. The possible applications are manifold, ranging from alarm systems to intelligent guidance systems, passenger recognition in car seats, air conditioning control in interior lining and smart wallpaper with software-defined light switches.

1 Introduction

Many promising technologies are emerging in the area of intelligent textile materials like electrically conductive yarns or pressure sensitive fabrics [1, 2]. State of the art feature sizes of integrated circuits allow for powerful and yet small and cost-effective microelectronic devices. Many interesting applications in the field of technical textiles arise by merging micro-systems and textile fabric structures [3]: pressure sensors in floor coverings for alarm systems or motion detection (person tracking), indicator lamps in floor or wall coverings for guidance systems in public buildings, distributed sensor networks for detection of defects in textile concrete constructions, and many more.

Approaching the given task of electronics integrated into large areas the following questions arise: How can we exploit the functionality of all the integrated microprocessors, sensors and light emitting diodes? What happens, if the smart fabric is cut to fit as e.g. a functional floor covering of an arbitrarily shaped room? Will a single destroyed or defective module or wire lead to a complete failure of the function of the smart textile system? To address these problems we decided to use a self-organizing and fault-tolerant architecture for the integrated sensor network. Research on self-organizing systems is presently pursued extensively, worldwide [4]. Most sensor networks are based on wireless data transmission [5, 6, 7, 8].

Several years ago, we developed ADNOS (algorithmic device network organization system) for building a self-organizing and fault-tolerant wired peer-to-peer network for large sensor and actuator areas [9]. Our first demonstrator of this technology is the "Thinking Carpet" described below.

In a previous work Paradiso et al. [10] described a "Magic Carpet", where an array of piezoelectric wires and Doppler radar motion sensors are used to track the motions of a performer in musical installations. Orr and Abowd [11] use a single pressure sensitive tile within the "Smart Floor" for identification of persons by their characteristic foot pressure profile during walking. In contrast, by using ADNOS we achieve a homogenous touch-sensitive floor, which is capable of self-installation and automatically circumvents defective areas of the sensor network. Moreover, through the localization of the sensor signals, speed and direction of the movement can be analysed and trigger software-defined events in the PC application.

2 Smart Textile Concept

Figure 1 shows a schematic of the smart textile. Identical modules are connected to each of their four neighbours by interwoven conductive filaments in a fabric used as data or power supply lines. One of the modules is connected to the PC and used as the portal. As depicted, several defects may occur within such a wired peer-to-peer sensor

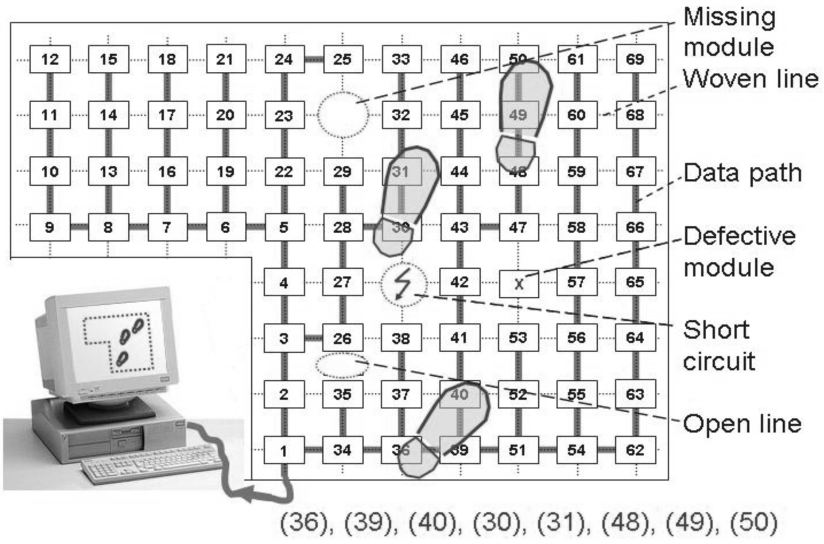


Fig. 1. Schematic of the wired network within the "smart textile", showing the automatically numbered modules and routed data paths; the indicated possible defects can be handled by ADNOS; sensor data are sent to the PC via the portal

network during fabrication or use: open lines, destroyed or missing modules and electrical short circuits. For a high yield in a reel-to-reel production and robust functionality fault-tolerance to all those defects is required. In addition, the demand for simple and low-cost installation requires that the smart textile can be cut into irregular shape to fit into any given room.

3 Textile Integration of Microelectronics

The smart textile (Fig. 2) is based on a polyester fabric with interwoven silver-coated copper wires. The diameter of each wire is 70 µm. Three of these wires are spun to one cord. For redundancy and low line resistance four cords are used for each line. The resulting line resistance is 0.4 Ω/m. The pitch of the woven pattern is 20 cm in weft and warp directions, respectively. The width of the textile is six pitches or 120 cm. Each ADNOS module is connected to its four neighbours by two supply and two data lines. Two additional lines per module are used as sensor lines for the touch sensor. To achieve a larger sensitive area of the touch sensor, we embroider a meander-shaped wire. The modules are connected at the crossover points of the conductive lines in a single step using anisotropic-conductive adhesive. The achieved contact resistance is 7 mΩ/contact. In contrast to previous work [1], we use non-insulated interwoven wires for reducing the number of process steps and therefore cost for the electronic/textile interconnect. At the crossover areas the conductive weft and warp fibres are removed to eliminate the electrical shorts of the interwoven conductive wires. The modules are encapsulated before mounting to reduce mechanical stress of the devices on the PCB.

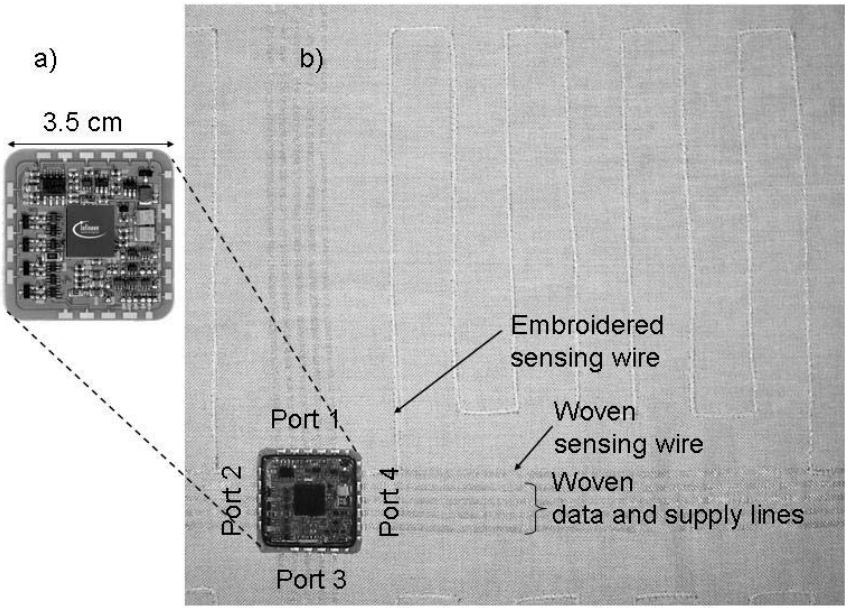


Fig. 2. Microprocessor module with a capacitive sensor integrated into a fabric with interwoven silver-plated copper wires. The embroidered meander-shaped wires form the touch-sensitive area for the capacitive sensor

4 ADNOS Peer-to-Peer Network

Figure 3 shows the block diagram of the ADNOS module. Each ADNOS module has four UARTs as ports to the connected neighbours and the input of the capacitive touch sensor. The power supply of the textile uses a voltage of 12 V in order to decrease the distribution losses. It is reduced to 3.3 V at the module by a switched power supply. For the demonstrator we use a commercially available 16 Bit microcontroller.18 KB Flash and 4 kB RAM are sufficient for our system. The modules are active during data transfer, only. Their power consumption is approximately 10 mA/module at 12 V in active and 6 mA/module in standby mode.

Within the network each module exchanges control messages with its four nearest neighbours and controls and drives a specific region. No prior knowledge about their position within the grid is used. The control data are fed into the network by a functional block referred to as portal, which is connected to an arbitrary module at the edge of the array.

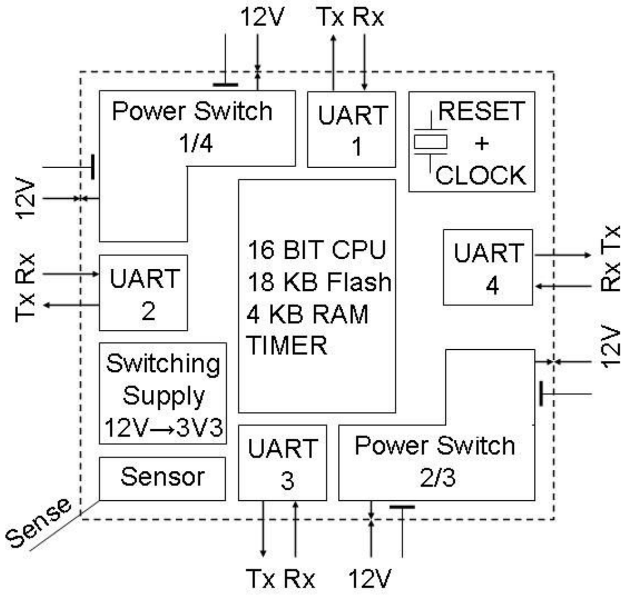


Fig. 3. Block diagram of the ADNOS module

A set-up phase is started at each module as soon as it is connected to the power-supply: During this first phase called "power routing", short circuits within the network are detected and affected branches switched off. The power switches at each module are able to switch up to 1.25 A. Their resistance is below 150 mΩ. In larger networks we prefer to use several power inputs uniformly distributed along the edges of the network.

Next, the self-organization of the network according to the ADNOS rules is started by the PC. Every module computes the positions of its neighbours, using its own temporary position set to (0,0) at the beginning. The portal starts this process by feeding the absolute position to the first module. In the next phase the minimum distance to the portal is computed by each module sending its own estimated distance to its neighbours. On reception of such messages, each module selects the neighbour with the smallest distance d and sets its own distance to d+1. Based on the previously computed data, routes are generated, that will later carry the data stream (Fig. 1). The

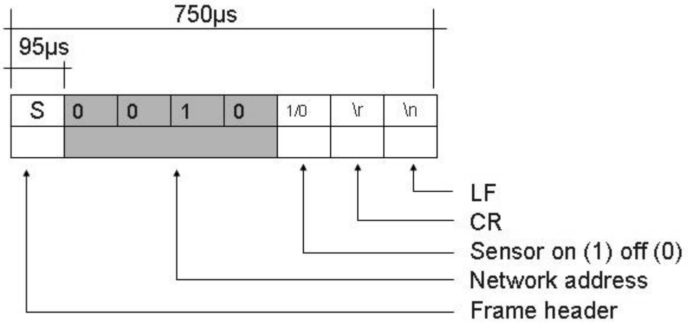


Fig. 4. ASCII-based message format used for communication within the network. This example shows the sensor frame

portal then starts an automatic numbering based on the established routes and the calculated throughput through each module. This enables the modules to determine a unique address number. The generated address number format yields sufficient routing information for each module. Except for the address numbers of its direct neighbours, no additional routing tables are needed inside the network. If new defects occur, the self-organizing routine can be repeated and new routing paths emerge in the network. The ASCII-based message format used for the communication within the network is shown in Figure 4.

The ADNOS algorithms are embedded in one of three layers of software abstraction (Figure 5). It establishes a peer-to-peer communication scheme, which is used to transfer data between the modules and between the network and the "smart textile" monitor application on the portal PC, respectively. The four ports of each module have different priorities. At simultaneous reception on two ports (messages received within 8 µs), the port with highest priority will accept data first. To avoid an unbalanced response of the network, the priority of ports is rotated at regular intervals. Below the ADNOS layer the protocol layer is responsible for communication between neighbouring modules. In case of data collision, reception and transmission is performed in full duplex mode. The board layer takes care of hardware control and physical connection between neighbouring modules.

The sensor data from the network are transmitted to the "smart textile" monitor application. We use an RS232 interface at a data rate of 115200 bps. The customized features are defined within the monitor application, e.g. processing and evaluation of sensed data or control of light-emitting diodes.

The PC user interface of the "smart textile" monitor is shown in Figure 6. All recognised modules, the functional connections between the modules and the established data paths are depicted in the left area of the screen. Sensor events are shown as highlighted dots at the connected module. The pattern shown in Figure 6 was produced by a person walking from the lower left to the upper right of the smart textile. All information gained within the network during the self-organization like i.e. coordinates, address, throughput, distance to the portal can be depicted on demand on the screen.

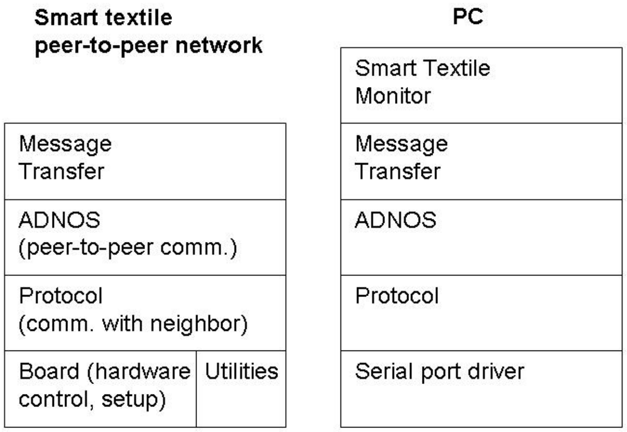
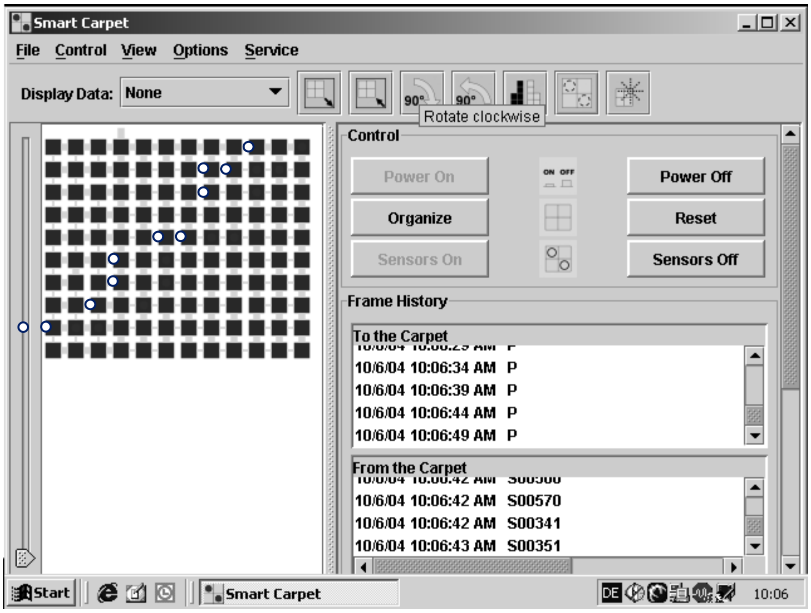

Smart textile peer-to-peer network
PC
Smart Textile Monitor
Message Transfer
Message Transfer
ADNOS (peer-to-peer comm.)
ADNOS
Protocol (comm. with neighbor)
Protocol
Board (hardware control, setup)
Utilities
Serial port driver

Fig. 5. Software scheme for the "smart textile" peer-to-peer network and PC



```
10/6/04 10:06:34 AM  P
10/6/04 10:06:39 AM  P
10/6/04 10:06:44 AM  P
10/6/04 10:06:49 AM  P
```

```
10/6/04 10:06:42 AM  S00570
10/6/04 10:06:42 AM  S00341
10/6/04 10:06:43 AM  S00351
```

Fig. 6. Screenshot of the ADNOS user interface on the PC featuring a "smart textile" network with 120 integrated modules (left side). The white dots indicate the sensor signals produced by a person walking from the lower left to the upper right edge of the textile

5 Results

To illustrate the functionality of the ADNOS system we fabricated a demo board with twelve modules. Figure 7a) shows a photo of the demo board with pluggable modules and three screen shots of the ADNOS user interface on the PC (b-d). The modules are depicted as dark squares. The broad lines are established data paths., thin light lines are identified connections not used as data paths. Figure 7b) shows the screen shot of the network with all modules plugged in. In Figure 7c) one module in the middle of the network has been removed as it is shown in Fig. 7a) to simulate a defective module. ADNOS has recognized the failure (marked dark) and shows, that all modules connected to the same data paths are not longer contributing to the functionality of the network (the colour changed to light). After initializing the reorganization a new data path is found surrounding the missing module (Fig. 7d). All remaining modules are fully functional again.

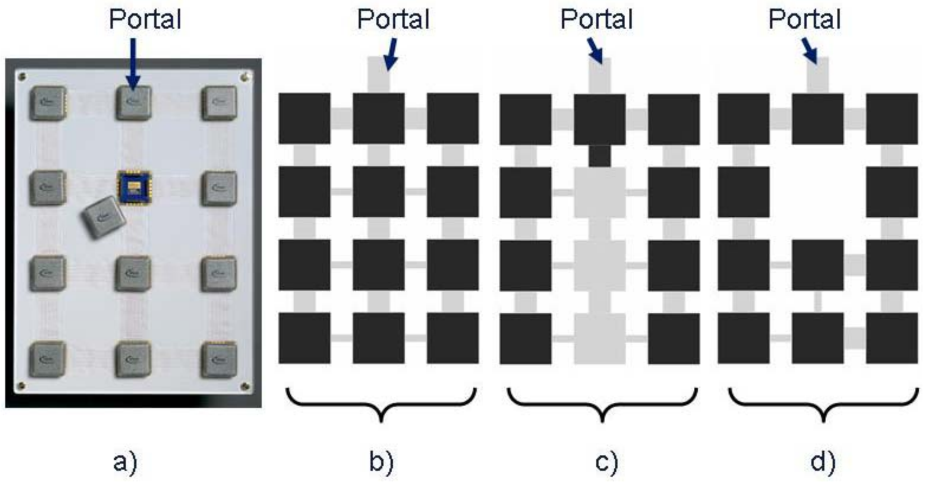


Fig. 7. ADNOS demo board with twelve pluggable modules (a) and three screen shots of the ADNOS user interface on the PC (b-d), the dark squares symbolize the modules, the broad grey lines are data paths, the thin grey lines are identified connections. b) original network, c) one module removed, c) failure (marked dark) recognized by ADNOS, light grey modules connected to the same data paths are non-functional, d) after reorganization

The maximum size of the network, using a single connection for the power supply can be calculated from the measured results derived from our smart textile demonstrator. The cumulated resistances in the network are 0.4 Ω/m for the interwoven copper wires and 7 mΩ/contact for the interconnect to the interwoven copper wires. The on-resistance of the FET switches is 150 mΩ. The standby current of one module is 6 mA at 12 V. For the supply voltage we use a switched power supply with 85 % efficiency at 12 V. The network is fully functional down to 8 V. Figure 8 shows the voltage drop of the supply voltage across one row of modules. The pitch of the modules is 20 cm and 50 cm, respectively. At the usual width of a carpet of 4 m the voltage drop is 1.1 V at a pitch of 20 cm (20 modules) and 0.25 V at a pitch of 50 cm (8 modules).

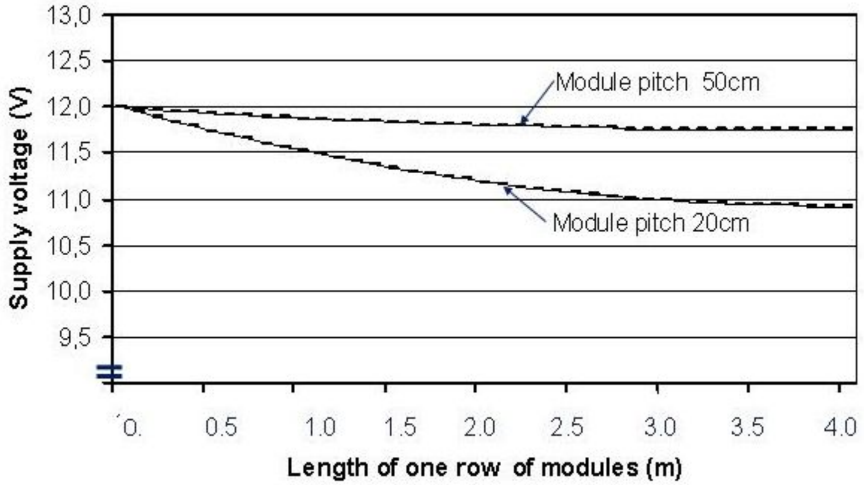


Fig. 8. Calculated voltage drop of the supply voltage across one row of modules integrated in the smart textile versus the length of the row. The pitch of the modules is 20 cm and 50 cm, respectively

Figure 9 shows the measurement of the delay time in a network with 60 members (6x10) as a function of the number of hops from a stimulated sensor of a module to the portal. We find an average time of 1.5 ms per hop, independent of the distance from the PC portal. An additional delay of approx. 1.5 ms will occur only, if data collisions occur at one module. The data traffic within a large area is relatively low, therefore the probability for simultaneous data reception is very low.

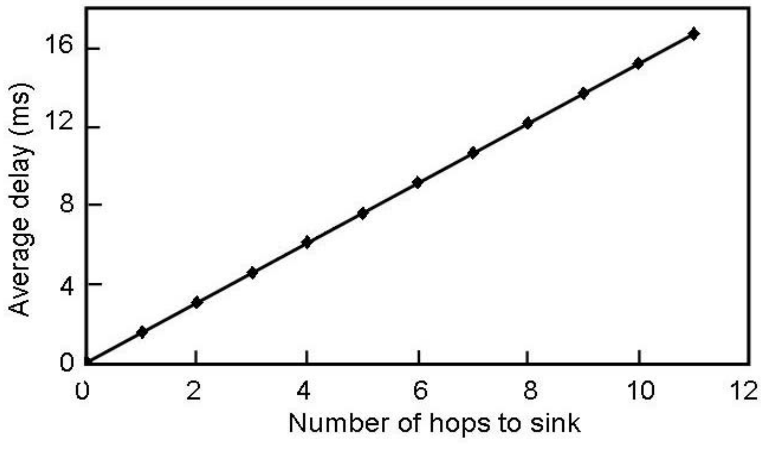


Fig. 9. The measured average delay time of the sensor signals is 1.5 ms and independently from the distance to the portal

6 “Thinking Carpet” Prototype

The integration technique of the smart textile in a carpet was developed in cooperation with the carpet manufacturer Vorwerk-Teppich and presented at the Orgatech 2004 in Cologne, Germany.

Using the technology described above, four “thinking carpet” prototypes were fabricated with 60 integrated modules and a size of 120 cm x 220 cm, each. The smart textile was cold-laminated between two layers of textile to reduce the mechanical stress and equalize the height of the modules (2 mm). The top layer is a tufted carpet combined with a 1000 gram fleece as backside. The power consumption of each carpet is 8.0 Watts. For the Orgatech prototype three carpets were connected to each other, therefore forming one single network. Figure 10 shows a photograph of the “thinking carpet” taken at the Orgatech. The touch sensitive floor is the darker rectangle in the photograph. The large display at the back plane shows that all 180 integrated modules are recognized. At a closer look at Figure 10 one can see, that there are open lines between several modules. However, due to the ADNOS self-organization new data paths were established surrounding the defects. All modules and sensors are fully functional. The light dots shown at the display are sensor signals produced by the dancer.

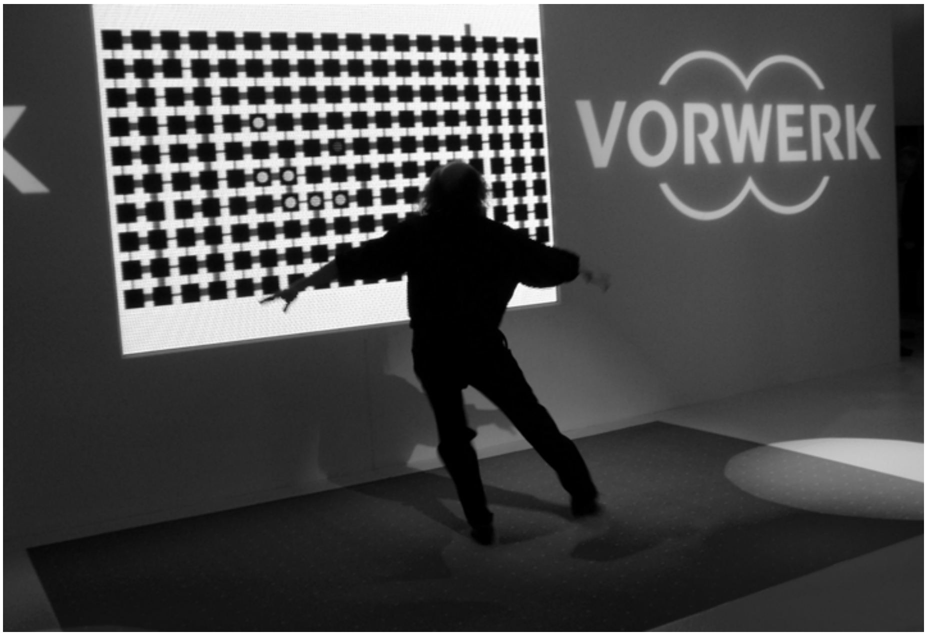


Fig. 10. “Thinking carpet installation” at the Orgatech 2005 (darker rectangle at the floor), featuring 180 ADNOS modules with capacitive sensor areas. The display on the back plane shows the recognised modules. The light dots indicate sensor signals produced by the dancer

The integrated capacitive touch sensor is fully functional through an insulating layer up to 30 mm thickness. Therefore a lot of materials like wood, stone, concrete, ceramic or glass can be used as flooring or as a protecting layer on top of the smart textile.

7 Application Scenarios

The integrated capacitive touch sensor remains fully functional even through an insulating layer of up to 30 mm thickness. Therefore a lot of materials like wood, stone, concrete, ceramic or glass can be used as flooring or as a protecting layer upon the smart textile.

The PC application software can be adapted to the specific functions the smart textile should support. The customized features are defined within this application, e.g. how the sensed data are processed and evaluated, or how light-emitting diodes are controlled. A wide range of new applications is opened up by the smart textiles, featuring the self-organizing and fault-tolerant microelectronic integration technique.

Convincing examples for applications are smart textiles working as alarm systems in the flooring of private or public buildings, in tents or truck tarps. Textile reinforced concrete with an integrated ADNOS network could automatically detect cracks after an earthquake.

An interesting application is the smart floor in apartments for elderly or handicapped people. Different functions can be triggered, using data mining on the sensed data: The light will be switched on automatically if a person enters the room or steps out of bed. If a person doesn't leave the bed for an unusual long period, a nurse will be called. Doors will open and close automatically if the person moves towards it. If a person falls down on the floor and doesn't move afterwards an emergency call will be activated. Figure 11 (left side) shows the sensor signals derived from the smart textile with a person, lying on the floor as depicted on the right picture. Such functionality would give elderly or handicapped persons a chance to live a self-determined life, without running an intolerably higher risk in case of an emergency.

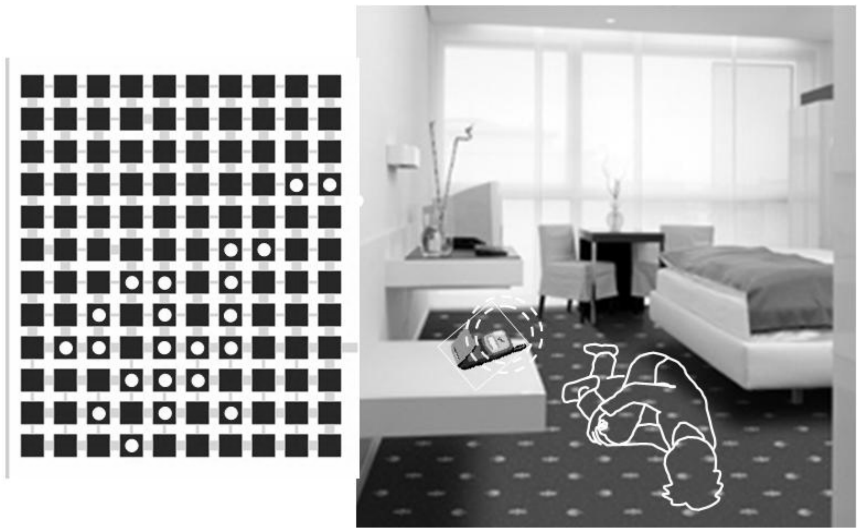


Fig. 11. Screen shot of the sensor signals (left) derived from a person that lies on the smart textile prototype like depicted in the right picture

Security applications in public or private building are of increasing interest. Statistical evaluation of the sensed data derived from the smart floor will distinguish "normal" behaviour of airport users from unusual behaviour of possible terrorists. Security areas will be defined within the software application. Having information about speed and direction of movements, footsteps starting from the window side of a building will trigger a burglar alarm.

In future, the functionality of the ADNOS module will be integrated in a single silicon chip with an estimated chip size of approximately 7 mm² using a 180 nm CMOS process. A density of four modules per m² seems sufficient for smart floor applications, when using larger sensor areas. The range of suitable and available sensors and actuators is wide-spread: sensors are available for various parameters such as pressure, temperature, humidity, smoke, gas, and sound. Integrated light-emitting diodes will give the possibility to build large-area displays, thus achieving intelligent guidance systems or flexible displays.

References

[1] Jung, S., Lauterbach, C., Strasser, M., Weber, W.: Enabling Technologies for Disappearing Electronics in Smart Textiles, Proceedings of International Solid-State Circuits Conference, ISSCC, San Francisco, CA, USA, 2003

[2] Post, E. R., Orth, M., Russo, P. R., Gershenfeld, N.: E-broidery Design and fabrication of textile-based computing, IBM SYSTEMS Journal, Vol. 39, No. 3&4, 2000, pp. 840-860

[3] Marculescu, D., Marculescu, R., Zamora, N. H., Stanley-Marbell, P., Khosla, P. K., Park, S., Jayaraman, S., Jung, S., Lauterbach, C., Weber, W., Kirstein, T., Cottet, D., Grzyb, J., Tröster, G., Jones, M., Martin, T., Nahkad, Z.: Electronic Textiles: A Platform for Pervasive Computing, Proceedings of the IEEE, Vol. 91, No. 12, 2003, pp. 1995–2018

[4] Di Marzo Serugendo, G., Karageorgos, A., Rana, O. F., Zambonelli, F.: Engineering Self-Organising Systems, ISBN 3-540-21201-9, Springer-Verlag Berlin Heidelberg, 2004

[5] Hester, L., Huang, Y., Nadric, O., Allen A., Chen, P.: NeuRon TM netform: a self-organizing wireless sensor network, Proceedings of Eleventh International Conference on Computer Communications and networks, Miami, USA, 2002,, pp. 364–369

[6] Basheer, M. R., Rao, V., Derriso, M.: Self organizing wireless sensor networks for struc tural health monitoring, Proceedings of the SPIE, 2003, pp. 515–525

[7] Clare, L. P., Pottie, G. J., Agre, J. R.: Self-organizing distributed sensor networks, Proceedings of the SPIE, 1999, pp. 229–237

[8] Guo, Y., Poulton, G., Valencia, P., James, G.: Designing Self-Assembly for 2-Dimensional Building Blocks, Di Marzo Serugendo et al.: AAMAS 2003 Ws ESOA, LNAI 2977, Springer-Verlag Berlin Heidelberg, 2004,, pp. 75–89

[9] Sturm, T. F., Jung, S., Stromberg, G., Stöhr, A.: A Novel Fault-Tolerant Architecture for Self-organizing Display and Sensor Arrays, 2002 SID Symposium Digest of Technical Papers, Volume XXXIII, Number II, 2002,, pp. 1316–1319

[10] Paradiso, J., Abler, C., Hsiao, K., Reynolds, M.: The Magic Carpet: Physical Sensing for Immersive Environments, Proceedings of the CHI '97, Conference on Human Factors in Computing Systems, Extended Abstracts, ACM Press, NY, 1997,, pp. 277–278

[11] Orr, R. J., Abowd, G. D., The Smart Floor: A Mechanism for Natural User Identification and Tracking, Proceedings of the CHI '00, Conference on Human Factors in Computing Systems, The Hague, Netherlands, 2000

[12] Vorwerk-Teppich: "Thinking carpet", für das Büro von morgen, http://www.vorwerk-carpet.com/sc/vorwerk/template/thinking_carpet.html

Applying Distributed Adaptive Optimization to Digital Car Body Development

Sven A. Brueckner[1] and Richard Gerth[2]

[1] Altarum Institute,
3520 Green Court, Ann Arbor, MI 48105, USA
sven.brueckner@altarum.org
[2] Center for Automotive Research,
1000 Victors Way, Ann Arbor, MI 48104, USA
rgerth@cargroup.org

Abstract. Companies in today's automotive industry are under immense competitive pressure to reduce the length of their product development cycle from initial concept to begin of high-volume manufacturing. A very costly and immensely knowledge-intensive step in this process is the creation of tools and dies required to manufacture a car body of a specified design. This paper presents a novel architecture for a decision support system that streamlines the development process through the integration of a virtual assembly simulation, problem identification, and solution generation and evaluation. Following the virtual functional build process, our architecture deploys a number of multi-agent systems to provide system functionality, such as problem knowledge retrieval or solution generation and evaluation.

1 Introduction

Today's fierce competition in the automotive industry pressures companies to find ways to drastically reduce time-to-market while increasing the quality of new vehicles. A key element in the launch process is the body development and manufacturing validation. This step is often a bottleneck and the most costly and thus limiting aspect of the vehicle launch preparation. The high cost in terms of duration and money is due to the intense human involvement in the process as current practice relies heavily on human knowledge and experience with very limited means of evaluating proposed solutions other than actual physical implementation.

Two recent technological advances provide essential building blocks that allow us to move from experience-based to data-driven body development. First, we now have the ability to efficiently create, transfer and store high-resolution digital scans of 3-dimensional parts; and secondly, the integration of Finite Element Analysis (FEA) and dimensional models enables us to predict residual stresses in a functional build assembly. Thus, at this point, suppliers can produce parts using prototype tools and dies and submit scans of these parts to the OEM. The scans of the parts are then assembled in simulation and the resulting sub-assembly or complete car body may be compared with the design intent.

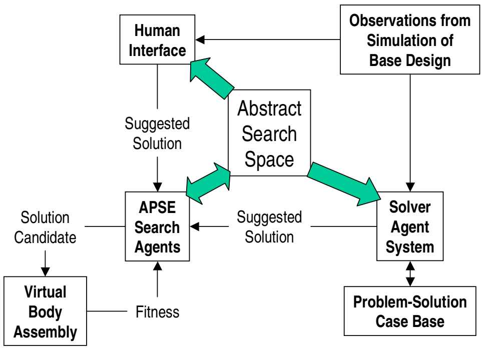


Fig. 1. DBDS performs a parallel heuristic search with human and case-based guidance

To close the loop in the virtual functional build process, we develop a decision support system (DBDS – Digital Body Development System) that analyzes the virtual product, identifies symptoms of underlying problems in the current design, and proposes and evaluates alternative solutions to the human design team based on past experience and heuristic search. The launch team then has the option of proposing additional solution alternatives or choosing a solution for implementation.

DBDS treats the generation of solutions to problems identified in the current design as a search problem in the high-dimensional space of modifications to the design guided by a fitness function. Any point in this abstract search space is a set of parameterized changes to the current design. Computing the fitness of such a set of changes requires the application of these changes to the design, and the simulation and analysis of the resulting new design comparing it with the current design.

In [3] we present an experimental application of our agent-based Adaptive Parameter Search Environment (APSE), which performs a heuristic parallel search across an abstract space of input parameters to an arbitrary simulation model guided by a fitness function defined over metrics reported during the execution of the model. DBDS is an application and extension of APSE in which sets of design changes are treated as input parameters to the virtual assembly of a car body and in which the search is guided by the design intent of the functional build process.

Given the complexity and massiveness of the search space that DBDS must explore in a given optimization run, we enhance the heuristic of the APSE Search agents to include prior experience and domain knowledge accessible in a problem-solution case base and we enable the human design team to suggest alternative solutions to the search process (Figure 1). We implement a Solver, a multi-agent system that interacts indirectly with the APSE Search agents and that seeks to retrieve solution points (sets of design changes) from a case base. The retrieval is guided by the problem symptoms observed in the execution of the current design and by the fitness of solutions that have been evaluated already by the Search agents.

The remainder of this paper is structured as follows. Section 2 discusses the current practice of auto body development in more detail. In Section 3 we present the DBDS component architecture, which closes the loop in the virtual functional build. Section 4 discusses the adaptation of the APSE Search agents to the decision support problem and Section 5 outlines our swarming approach to solution retrieval. We conclude in Section 6.

2 Current Practice in Body Development

Detailed engineering design of individual parts and components begins after "design freeze". This typically includes a finite element analysis (FEA) of the nominal design to examine stresses, vibration, crash testing, etc., as well as a tolerance analysis to determine how the components will fit. This latter analysis often involves identifying designs that are sensitive to variation and making the design more robust by changing and redesigning parts to reduce geometric effects. Once the individual part design is set, it is released for "tooling" (tool release – i.e. the process of constructing the stamping dies), and the functional build process begins.

Functional build is a critical process in launching a vehicle (see Figure 2), whereby individual prototype parts are stamped and then sent to a central location to be assembled into a prototype vehicle body [1]. Since production tooling is often not yet available, the body is fastened with screws and rivets, hence it is called a "screwbody." The screwbody is examined by experienced experts who must decide whether gaps and interference conditions between individual parts are sufficient to warrant changing the dies, the welding tooling, clamp locations, etc. If it is decided that a change is warranted, then the dies may have to be returned to the supplier to be changed. If a change is not warranted, then the specifications may be changed to match the part shape. This usually involves a uni- or bi-directional opening of the part tolerances. The process is then repeated after the changes have been implemented. It is not uncommon to have three or more functional build evaluation bodies during a vehicle launch, which is costly and time consuming. However, each evaluation is based on a different generation of tooling, so very little information is gleaned on the effects of process variation to the integrity of the entire body.

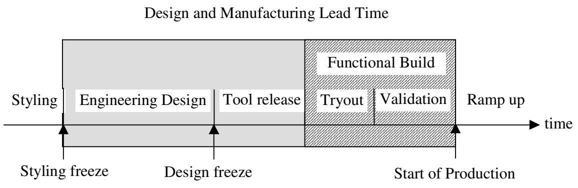


Fig. 2. Vehicle Launch Process

Two technological trends aimed at improving the dimensional integrity and performance (NVH – or noise, vibration and harshness) of the body are 1) the integration of Finite Element Analysis (FEA) and dimensional models and 2) the scanning of fabricated parts and assemblies for comparison of actual builds with the design intent.

The integration of FEA and dimensional models is significant in that it allows the prediction of residual stress in a functional build assembly. Conceptually, the parts are assembled in the software. Any interference or gap conditions will be accommodated by the assumption that sheet metal is a compliant part. Weld points are identified and the parts are forced into full contact at those points. These points are held as boundary conditions. Then the FEA program minimizes the stress in the assembly by changing the shape of the part according to the boundary conditions.

The recent progress of combining FEA and dimensional models significantly advances the science for understanding the complex interactions between sheet metal parts and the joining processes (usually spot welding). The effects of interference and gap conditions between two mating parts are evaluated based on the amount of part "compliance" that can be expected. Compliance (the bending of parts as they are joined together) can be predicted using FEA, which is also used to predict residual stress in the assembly. The dimensional model can expand that understanding over the expected variation of the fabricating and assembly processes. Together, these two tools can quantify manufacturing capability (fabrication and assembly) and produce a distribution of residual stress as well as dimensional measures in the body.

3 Simulation-Based Decision Support

The Digital Body Development System (DBDS) provides continuous support for the vehicle launch team along the entire iterative functional build process. A single iteration starts at a base design, which comprises scans for all parts that have been produced at this point and CAD-nominals for the remaining parts. The base design also specifies the assembly process as it is currently planned.

In a first step, the base design is "executed" by simulating the virtual assembly of the parts and pre-defined measurements are taken on the resulting product. The design process is completed, if the results meet the design intent. Otherwise, DBDS interprets the output of the measurements as symptoms of underlying problems and generates and evaluates solution alternatives (changes to the base design). It may also invite human engineers to suggest additional solutions. Eventually, the launch team will settle on a solution and implement the corresponding design changes (i.e., change tools and dies, make and scan new parts) to arrive at a new base design.

In today's practice of body development, a large team of experts with diverse background and experience analyze the current design as it manifests itself in the screwbody. Based on their domain knowledge and past experience, individual experts suggest solution alternatives and then discuss their potential merit until the team agrees on a solution. The whole solution generation and selection process is dominated by human knowledge and experience and solutions are chosen or discarded mainly based on hypotheses rather than evidence. Alternatively, DBDS explores the space of possible solutions to the currently observed problems and evaluates each solution alternative by simulating the design that results as the changes proposed by the solution are applied to the current base design. Thus, solutions that DBDS suggests to the launch team are based on evidence provided by the simulation rather than hypotheses.

The simulation-based improvement of a given base design using a heuristic search and evaluation process may be applied to domains other than car body development. To facilitate such a transfer later on, we specify a generic module architecture that makes the specifics of the domain transparent to the optimization process.

The DBDS decision support system has seven modules (Figure 3). The User Interface (UI) module manages the interaction of the system with the human design team. The Solution Generation and Evaluation (SGE) module proposes alternative solutions to solve problems with the base design and evaluates them for their quality and cost. The Change Cost Estimation (CCE) module estimates the cost of actually implementing a particular solution as a change to the base design. The Solution Implementation

(SI) module translates a proposed set of changes to the base design into a valid modified design that can be simulated by the VASE. The Virtual Assembly and Simulation Engine (VASE) simulates the "execution" of a given design by virtually assembling parts according to a process description. The Data Preparation and Repository Module (DPRM) manages the large amounts of data generated and used by the DBDS. The Process Controller (PC) module integrates the other modules and manages the data and process flow among them.

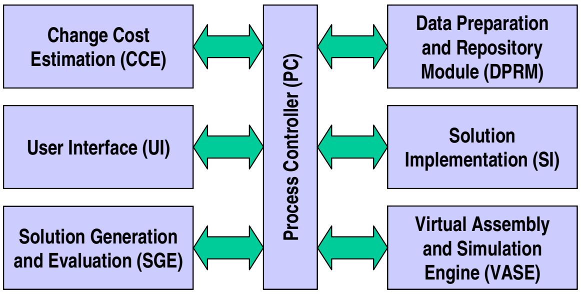

Change Cost Estimation (CCE)
User Interface (UI)
Solution Generation and Evaluation (SGE)
Process Controller (PC)
Data Preparation and Repository Module (DPRM)
Solution Implementation (SI)
Virtual Assembly and Simulation Engine (VASE)

Fig. 3. The Generic DBDS Module Architecture

Figure 4 illustrates the high-level (black) and low-level (white) process loops facilitated by the PC. At the high level, the user triggers the improvement of a base design, which requires its execution (VASE) and analysis and optimization (SGE). The optimization process generates alternative solutions, which are evaluated (lower-level loops) for their performance (SI, VASE) and cost (CCE).

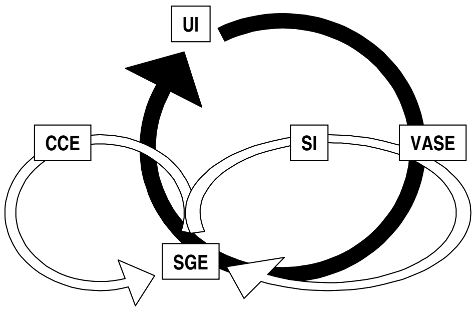


Fig. 4. Inner (white) and Outer (black) DBDS Loops

4 Heuristic Search for Design Changes

In [3] we present APSE – a multi-agent system that performs a distributed heuristic search through the space of input parameters of a black-box simulation model to find a configuration that maximizes a fitness function defined over observed metrics on the simulation. The APSE Search agents collaboratively explore the space of potential solutions (model parameters) and evaluate them through successive simulation runs. Using a Particle Swarm Optimization (PSO) algorithm [6] combined with probabilistic local hill climbing, the agents coordinate their activity so that computing resources (simulation runs) are focused on exploring the most promising regions of the search space.

The Solution Generation and Evaluation (SGE) module of the DBDS hosts an APSE Search agent population, whose task it is to explore the space of possible changes to the base design for improvements that reduce or remove the problems observed in its execution. Thus, we treat changes to the base design as input parameters to a black-box simulation and define a fitness function for the search process that measures the degree to which the now modified design meets the design intent.

DBDS is an enhancement of the APSE architecture. While Search agents in APSE are guided only by the fitness of the currently known solution candidates (points in the abstract search space), DBDS provides two additional sources of guidance for the distributed search (see Figure 1). The first source of solution candidates is the human design team. At any point during the search process, human experts may look at the problem symptoms and the solutions DBDS has explored so far and suggest another solution to the system. Solutions may also be suggested by the Solver, a multi-agent system that seeks to match the problem symptoms to the descriptor of solution cases recorded in a case base (see Section 5).

We integrate these two additional sources of creativity into the search process by enhancing the APSE Search agents' behavior. In APSE, an agent explores the search space through a series of short-range moves that are guided by hill-climbing and PSO heuristics. In DBDS, a Search agent monitors the performance of its short-range movement heuristic (rate of improvement over time) and may decide to abandon its current region in search space through a long-range jump beyond the local correlation distance of the fitness function. The destination of the jump is a solution candidate provided by the human design team or the case-based Solver. Figure 5 illustrates the emerging agent trajectory in an abstract search space.

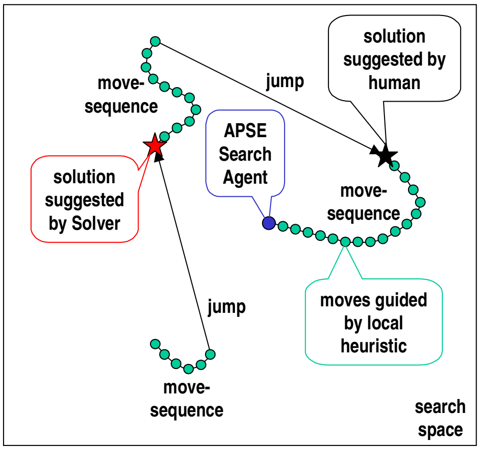

solution suggested by human
jump
move-sequence
APSE Search Agent
solution suggested by Solver
move-sequence
jump
moves guided by local heuristic
move-sequence
search space

Fig. 5. Agents move and jump through the search space guided by local heuristic, human input, and case knowledge

The generic distinction between a local improvement heuristic and global jumps to externally suggested solution candidates is open to other solution approaches. Just as DBDS currently implements a case-based approach to the solution of problems with the base design, other (e.g., rule-based, model-based, etc.) approaches could be implemented independently and feed into the decision process of the Search agents.

5 Swarming Case Retrieval

Today's car body development process heavily depends on human expert knowledge and experience. With DBDS we create a decision support system that has the ability to discover new solutions on its own through a heuristic search and evaluation in simulation, while at the same time utilizing and capturing human creativity and expertise to move from experience-based to data-driven design.

The SGE module of DBDS includes a dynamic Solver that analyzes problems with the base design as they manifest themselves in observable symptoms during the virtual assembly and that suggests solutions to these problems drawn from a set of problem-solution cases. We integrate the Solver with the heuristic search process by suggesting solution candidates to the APSE Search agents for their next long-range jumps and by modifying the case retrieval process based on the fitness of the solutions that have already been explored (Figure 6).

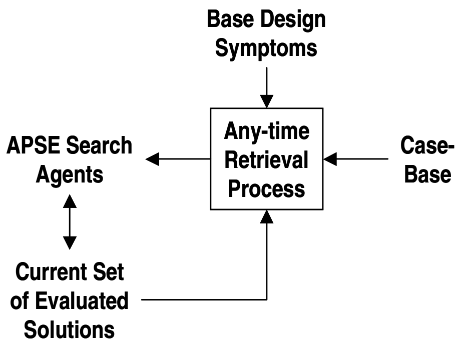

Base Design Symptoms
Any-time Retrieval Process
APSE Search Agents
Case-Base
Current Set of Evaluated Solutions

Fig. 6. The dynamic Solver modifies the solution candidates that it suggests to the Search agents based on the progress of the exploration of the search space

The ongoing asynchronous interaction with the Search agents and the continuous addition of fitness evaluations of new solution candidates requires a dynamic update of the case retrieval. Thus, we chose an agent-based any-time approach that continuously integrates changes in the external circumstances without having to restart its reasoning process from scratch.

In the following we discuss details of the operation of the Solver top down. First, we present the adaptive any-time process that manipulates the description of the current problem symptoms to provide a high-quality retrieval of high-performance solutions. Then, we specify the internal mechanics of the fine-grained agent system that drives the adaptive modification of the current problem description.

5.1 Linking Emergent Clustering and Spreading Activation Case Retrieval

The virtual assembly of the base design by the VASE module results in a large set of uniquely identified measurement points on the assembled car body that are either within or outside specified tolerances. Just as a fever, a cough and a runny nose are possible symptoms of an underlying viral infection, so are patterns of deviations at pre-defined measurement points on a (virtually) assembled car body symptoms of specific underlying problems (root causes) with the design.

Our dynamic Solver seeks to match the currently observed symptomatic patterns to those of problems encountered in the past, whose solution is recorded in the case base. We organize our case base into a simplified Case Retrieval Network (CRN) [7], which represents basic components of the problem description and the associated solution as individual nodes in a spreading activation network. The nodes representing problem components are called Information Entity (IE) nodes and a solution is stored in a so-called Case node. All IE nodes that describe the problem solved in a specific solution case are linked to the respective Case node through weighted relevance edges. The retrieval process first places an activation onto individual IE nodes depending on their match to the current problem symptoms and then propagates the activation through the relevance edges to the Case nodes. The relative activation of the individual Case nodes provides an ordering of the recorded solutions with respect to their relevance to the current problem.

Our goal is to abstract away from the specific locations and count of measurement points provided by the simulation by identifying symptomatic regions on the virtual car body that may be expressions of the same underlying problem. For instance, if a door is set slightly off-center into its frame, we may find several disconnected regions along the frame in which our pre-defined measurements are out of tolerance (e.g., gaps, interferences). To that end, the Solver executes a fine-grained multi-agent system that continuously rearranges measurement points into clusters that form components of the problem signature (Figure 7). The currently emerging problem signature is matched against past problems' signatures in the case base to provide a relevance measure of the available solutions. This relevance measure guides the selection of the next solution candidate upon request of an APSE Search agent. We select a case probabilistically, based on its current normalized relevance.

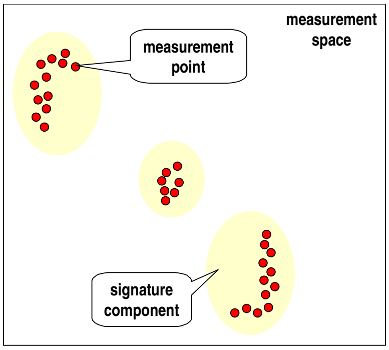


Fig. 7. Clustering of Measurement Points into Signature Components

The quality of the case retrieval process is high, if there are only one or very few cases with a significant probability to be selected. Otherwise, we may as well select a case randomly from the entire case base. We measure the current retrieval quality with the Case Selection Entropy (CSE) metric, which is the Shannon (Information) Entropy [11] of the case selection probabilities. The current CSE, resulting from the interaction of the current arrangement of measurement points with the Case Retrieval Network, may modify the behavior of the agents in the next clustering cycle. We have used similar entropy measures defined over the current preferences of an autonomous decision maker (here case selection) in previous projects [4, 9] to estimate the current information these preferences actually convey and to subsequently adapt the decision process if necessary.

Figure 8 illustrates the tight feedback loop (black) between the ongoing clustering of measurement points and the current case relevance ordering provided by the CRN. Through this feedback, the identified problem regions are modified to match past experience recorded in the case base more closely while maintaining a close tie with the actual problems observed in the simulation.

The clustering process is also influenced on a larger time scale by the observed performance of solutions that have been explored by the APSE Search agents (white loop in Figure 8). If a solution case is adopted by a Search agent in a long-range jump, DBDS evaluates the fitness of the changed car body design in terms of the reduction in problems compared to the base design and the estimated cost in implementing these changes. The fitness of all solution candidates proposed by the Solver is fed back through the Case Retrieval Network (activating case nodes and spreading to IE nodes) to attract the clustering mechanism away from or towards to specific arrangements.

5.2 Emergent Clustering

The output of the simulation is a cloud of values for predefined measurement points. Each point is associated with geometric coordinates on the car body, but it also carries additional context values, such as part features with which it is associated, assembly process steps that came in contact with the part, or the supplier providing the part. Thus, a measurement point is located in a high-dimensional space that combines the geometric and context dimensions. Through the additional context, we may associate points that are related in the process but not necessarily in geometry to the same signature component.

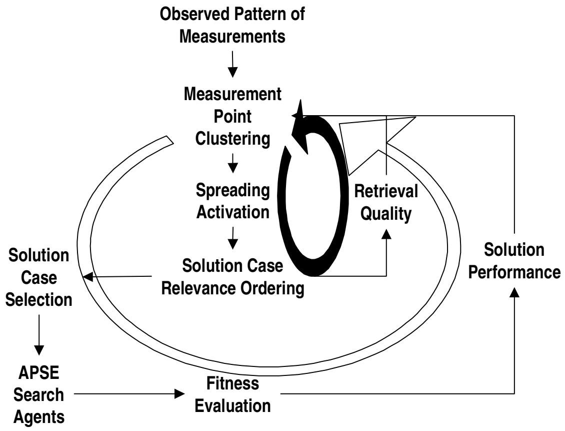


Fig. 8. Adaptive Case Retrieval Guided by Retrieval Quality and Solution Performance

Starting from the original locations of the measurement points, we seek to rearrange the points into arbitrary clusters while trying to keep each point close to its original location. As Figure 9 illustrates, there are a number of possible arrangements that meet these qualitative objectives, because we do not assume a particular number or size of clusters. We design our emergent clustering algorithm to potentially visit all these arrangements (with varying probability) and we use the feedback of the Case Selection Entropy metric and the currently known solution fitness to push the clustering system out of unfavorable configurations.

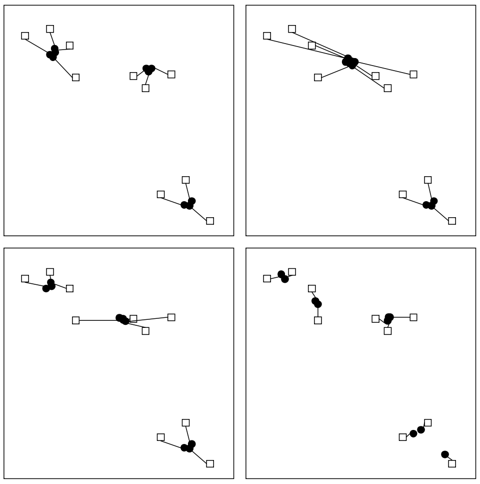


Fig. 9. Possible Cluster Arrangements (black) for the same Original Measurement Points (white)

Emergent any-time clustering is one of the prime examples of emerging functionality through stigmergic coordination in large-scale fine-grained multi-agent systems. Nest sorting [2], is an instance of emergent clustering observed in social insect systems. In this case, independent agents (ants) pick up or drop off passive objects with a dynamically computed probability. This behavior has been replicated in collective robotics (see for instance [5]). An alternative approach to clustering is to give the initiative to the objects themselves, which then reason about their current local arrangement and move about in space. We successfully applied this approach to create large-scale, self-organizing document bases [10] and we follow the approach in this application too.

In the emergent adaptive clustering algorithm, we assign each point an agent, which moves through the space of geometric locations and additional context. The sum of two dynamic force vectors, representing the two objectives in the rearrangement, determines the trajectory of an agent. The first force vector ("Home Force" in Figure 10) attracts the agent back to the original location of the measurement point. This force increases with distance. The second force vector is the sum of individual component vectors ("Cluster Force" in Figure 10), which each attract the agent to the location of another nearby agent. The strength of this force decreases with distance. The rates in which the forces change for changing distances are dynamic parameters of the system.

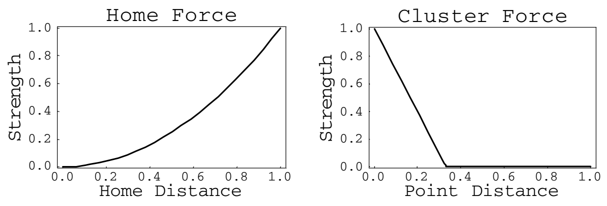


Fig. 10. Forces represent agent objectives in clustering

In each cycle, each agent calculates the home force and the cluster force vector from the position of the agents in the previous cycle. The vector sum of these two forces determines the direction into which the agent relocates in this step. The length of the step is the length of the combined vector, but limited to a relatively small step-length value (Figure 11).

If the force calculation algorithm in the agent were deterministic and used only constant scaling parameters, then the system would quickly stabilize on one arrangement that minimizes the "tension" among the objectives. To avoid unstable minima and to explore a variety of nearby cluster configurations, we add a small random component to the individual relocation calculation.

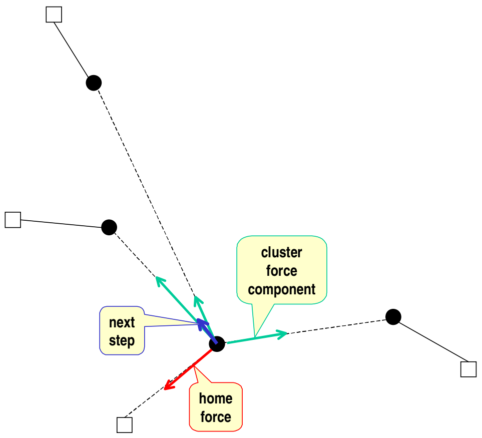


Fig. 11. Iterative Local Force Vector Calculation

We achieve qualitatively different cluster configurations through the feedback of the current retrieval quality and the solution performance, encoded in the Case Selection Entropy (CSE) and the fitness of solution cases (see Section 5.1).

The CSE metric offers a global evaluation of the value of the current point arrangement for the high-quality (non-random) retrieval of a solution from the case base, but it does not provide any guidance on how the arrangement should be changed to achieve a higher retrieval quality. Since higher CSE values correspond to low retrieval quality, we need to encourage exploration of new configurations over the exploitation of current clusters by increasing the impact of the random component in the agents' trajectory calculations.

The fitness of solution cases that have been explored by the APSE Search agents can be translated into directional guidance for the clustering agents. Before each cycle of the emergent clustering algorithm, we propagate the fitness of all cases (zero if not yet explored) backwards through the CRN to the IE nodes that represent regions of high point concentration (clusters) recorded with these past cases. Solution cases that led to an improvement in the design communicate a positive activation to their IE's while those that actually made the problem worse send a negative activation.

The positive or negative activation of IE's in the Case Retrieval Network translates to additional attractive or repulsive force components that steer points towards or away from regions in measurement space. We have used a similar back-propagation approach in CRN's to guide the interactive diagnosis of failures in computer hardware [8].

Table 1. DBDS Joint Venture Partners

Altarum Institute
American Tooling Center
Atlas Tool, Inc.
Autodie International
Center for Automotive Research
CogniTens Inc.
ComauPICO
UGS
Ford Motor Company
General Motors Corporation
Perceptron, Inc.
Riviera Tool Company
Sekely Industries
Thunder Bay Pattern Works

6 Conclusion

Car body development is the most costly step in the launch of a new vehicle and even small improvements of this process may yield high gains for the automotive industry. This paper presents the Digital Body Development System (DBDS) – a decision support system for the car body development team – which is an extension of the agent-based Adaptive Parameter Search Environment (APSE) presented in [3]. DBDS is based on a modular architecture, which makes the required activities of the evaluation of the fitness of solution candidates (simulation, cost estimate) transparent for the APSE Search agents exploring the space of changes to the current design of the car body.

The primary extension of APSE, besides its application to a highly complex domain, is the integration of external guidance into the local search heuristic of the agents. DBDS enhances the decision process of the individual agent, who now tracks the performance of the local improvement process (moves) and decides, whether to abandon its current region (jump) in favor of solution candidates suggested either by the human design team or a novel adaptive case-based Solver.

The case-based Solver is a complex adaptive system that interacts with the APSE Search agent population, providing it with solution candidates that may address currently observed design problems and adjusting its recommendations based on the fitness of the solutions that have been explored already. The Solver links a fine-grained agent system that continuously modifies the description of the current problem with a Case Retrieval Network that records solutions to past problems. The retrieval of solutions is refined by the agents' modification of the problem description, driven by the currently estimated quality of the case retrieval and the performance of selected cases.

The DBDS is the focus of an ongoing NIST/ATP-supported Joint Venture of more than a dozen automotive, software development and research companies and organizations (see Table 1). The architecture and algorithms reported in this paper are currently being implemented and tested and quantitative results from our first prototype will be forthcoming soon.

Acknowledgements

This work is supported by the DBDS Research Consortium, sponsored in part by a NIST/ATP grant (Project Number 00-00-5521). The views and conclusions in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Institute of Standards and Technology or the US Government.

References

[1] Auto/Steel Partnership (2000) Event-Based Functional Build: An Integrated Approach to Body Development - Complete Version, Auto/Steel Partnership, Southfield, Michigan, http://www.a-sp.org/publications.htm.

[2] E. Bonabeau, M. Dorigo, and G. Theraulaz. Swarm Intelligence: From Natural to Artificial Systems. New York, Oxford University Press, 1999.

[3] S. Brueckner and H.V.D. Parunak. Resource-Aware Exploration of the Emergent Dynamics of Simulated Systems. In Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems. Melbourne, Australia, 2003.

[4] S. Brueckner, H.V.D. Parunak. Information-Driven Phase Changes in Multi-Agent Coordination. In Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS 2003), 950-951. Melbourne, Australia, 2003.

[5] J. L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chretien. The Dynamics of Collective Sorting: Robot-Like Ants and Ant-Like Robots. In J. A. Meyer and S. W. Wilson, Editors, From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior, pages 356-365. MIT Press, Cambridge, MA, 1991.

[6] Kennedy, R. C. Eberhart, and Y. Shi. Swarm Intelligence. San Francisco, Morgan Kaufmann, 2001.

[7] Mario Lenz, Hans-Dieter Burkhard: Case Retrieval Nets: Basic Ideas and Extensions. in: G. Görz, S. Hölldobler (Eds.): KI-96: Advances in Artificial Intelligence, Springer Verlag, LNAI 1137, 1996.

[8] Mario Lenz, Hans-Dieter Burkhard, and Sven Brueckner. Applying Case Retrieval Nets to Diagnostic Tasks in Technical Domains. I. Smith and B. Faltings (eds.). Advances in Case-Based Reasoning. Springer Verlag, LNAI 1168. 1996.

[9] H. V. D. Parunak, S. Brueckner, R. Matthews, and J. Sauter. How to Calm Hyperactive Agents. In Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS 2003), 1092-1093. Melbourne, Australia, 2003.

[10] H. V. D. Parunak, P. Weinstein, S. Brueckner, J. Sauter. Hybrid Stigmergic Mechanisms for Information Extraction. In Proceedings of the Second International Workshop on Mathematics and Algorithms of Social Insects (MASI 2003). Atlanta, USA, 2003.

[11] C. E. Shannon and W. Weaver. The Mathematical Theory of Communication. Urbana, IL, University of Illinois, 1949.

Adaptive Service Placement Algorithms for Autonomous Service Networks

Sven Graupner, Artur Andrzejak, Vadim Kotov, and Holger Trinks

Hewlett-Packard Laboratories, Palo Alto, CA 94304, USA
{sven.graupner, artur.andrzejak, vadim.kotov, holger.trinkds}@hp.com

Abstract. Motivated by trends in the industry towards transforming IT in large integrated service networks, this paper describes algorithms for the adaptive placement of "services" (as abstractions of collections of applications) in networks of "servers" (as abstractions for locations where services can be hosted). Networks comprised of interacting services as the foundation is also a vision pronounced by the Grid [9]. Manageability and "self-operation" of Grids is highly desirable. We analyze the requirements for algorithms one specific problem: the service placement problem. We discuss algorithms that neither require central control nor complete information about the system state. Algorithms are performed on a distributed overlay structure which summarizes load conditions in the underlying service network. The presented algorithms fulfill tasks of making initial placement decisions as well as initiating rearrangements when imbalance is detected. Presented algorithms have different characteristics regarding the tradeoff between accuracy (or quality) of a placement decision and its timeliness within which a decision can be made determining responsiveness.

Introduction

In response to growing complexity and, as a result, the potential ineffectiveness and insufficient manageability of large-scale systems, new approaches to system design, use, and management are emerging:

- the aggregation and consolidation of system and application components into larger building blocks (services),
- systematic and standard ways of their integration and communication,
- sharing of distributed resources, and
- automated system management and operation control.

Two of the most prominent and practical examples are the concepts of the Utility Data Center [14] and the Grid [9]. The Utility Data Center (UDC) consolidates computing resources in order to significantly reduce deployment and operation costs. Grids are large networks of computing resources that can be transparently shared and utilized for solving complex tasks or providing computing services.

These two concepts can be combined into a concept of Virtual Data Centers [20] that consolidate resources of a federation of distributed Utility Data Centers into virtual resources that are shared using Grid–type mechanisms.

Grid. Grid computing emerged in scientific supercomputing in the early 1990's by making expensive compute resources available to external users. A software layer in form of Grid middleware provides the coordinated, transparent and secure access to shared resources across geographically distributed locations. Resource virtualization allows transparency and security. The software layer also provides "grid membership" of a machine or a device making its resources discoverable and allocatable to other entities in the system. Many Grid projects in research and industry are based on the Globus Toolkit [11], a widely used public domain software. Commercial Grid products are offered by IBM [15], Platform [26] and Sun [31]. An overview of Grid resource management systems can be found in [21].

Another source of the "grid trend" is the utility model of resources. Access to resources is aimed to be as simple and efficient as accessing power or other utilities. Resource markets are envisioned where resources used in information processing can be traded and exchanged as commodities. Resource commoditization also helps to overcome the diversity and complexity of IT landscapes making it attractive for both IT vendors and customers. So far, most integrated management systems are limited in regard to functioning in virtualized environments across organizational boundaries. Besides automated fail-over techniques in high-availability systems, management systems typically automate monitoring and information collection. Decisions are made by human operators interacting with the management system. Major service capacity adjustments imply manual involvement in making changes in hardware as well as in software. Systems need to be adjusted, re-installed and reconfigured.

Utility Data Center (UDC). A new type of data center infrastructures provides immediate support for these tasks. HP has developed the Utility Data Center (UDC) [14], [28]. Its capabilities allow a whole new approach to automate adjustment processes and by thus set the foundation for an automated service capacity-demand control system for a large service networks. This control system is based on a federation of geographically distributed data centers with capabilities providing immediate support for service demand and supply control.

Service Placement Problem. One of the hard problems in system management is the service placement problem, closely related to the (distributed) resource allocation problem. We assume that no global information about resource availability and service demand can be provided due to the scale and dynamism of large service networks. Placement algorithms thus need to deal with partial information, and yet provide good approximations of localized assignment solutions, and also need to be reactive such that decisions are made in appropriate time for an automated resource demand-supply control system.

In this paper we analyze the requirements for algorithms for the service placement problem in our context, discuss the trade-offs in the design of the algorithms, and formalize the objectives of a demand-supply control system. As a central part of our contribution, we propose several algorithms for the service placement problem under these requirements. Special focus of the presentation is directed on methods ensuring adaptability, fault-tolerance, and scalability.

It is noteworthy that each of the proposed algorithms assumes an infrastructure for actuating automated demand-supply control decisions. This architecture is defined by a demand and supply overlay meta-system. The meta-layer connects the various pools (data centers or parts of them) and allows algorithms to obtain a local view on conditions in the service or in the server layer. The architecture supports a variety of large-scale distributed systems including a federation of distributed Utility Data Centers.

The sequel is organized as follows. The following section is devoted to related work. Then we formalize the problem, present solution requirements and their limitations, and propose metrics for evaluation of placements of services. The first algorithm is based on the so-called Ant Colony Optimization. This class of algorithms originated from behavior studies of real ants and incorporates elements of machine learning by recording the best partial solution using a so-called "pheromone". Another approach is adopted from the coordination of mobile robots. It is called the Broadcast of Local Eligibility (BLE). We extend this method to provide better scalability than the original solution and suggest improvements in terms of communication costs by applying gossiping algorithms. The third algorithm combines a notion of intelligent agents which represent groups of services with P2P-based overlay networks.

Related Work

The basic service placement problem is closely related to the task allocation problem widely studied in the fields of parallel algorithms and clusters [7]. However, the additional solution requirements discussed below (in particular decentralization) make the solutions of this research unusable in our setting. Therefore we focus the related work discussion on approaches with inherent features of decentralization and adaptability, even if they provide different functionality than service placement.

In the area related of Grid technologies, the closest work to the one presented here is Messor [25], a decentralized load-balancing algorithm based on multi-agent systems. Similarly to the algorithm presented in this paper, it uses so-called Swarm Intelligence paradigm to distribute load in a pool of resources. As the infrastructure it uses the Anthill runtime environment [3], which combines a P2P platform and a multi-agent development environment in a single framework. Our algorithm differs from this approach by a more general model of service dependencies and by the mechanisms for disseminating of assignment evaluations.

The adaptation and self-organization of large distributed systems is also targeted by P2P systems research. In most of these systems mechanisms that automatically handle joining and leaving nodes (e.g. servers) are inherent parts of the design. This also applies to adaptation and fault-tolerance. Examples include Gnutella, Pastry, Tapestry, Chord and CAN [27]. Project OceanStore [22] exemplifies an application of a P2P-based indexing structure for resource management; another example of this kind is given in [2]. Most of the research in this area considers data management, while we focus here mainly on computational resources.

The most prominent project at the edge of self-organization and resource management is IBM's Autonomic Computing [17] aiming to create systems that are self-configuring, self-healing and self-optimizing. Related to this research is Océano [16].

Further related work concerning particular techniques used in our algorithms is discussed in the corresponding sections.

Problem Analysis and Solution Requirements

The basic service placement problem is defined as the problem of finding a mapping from a set of services to a set of servers. The term "service" is used in this paper as an abstraction for collections of program instances implementing applications such as customer relationship management systems in the enterprise IT domain or distributed applications in scientific computing. The term "server" generalizes the notion of a resource to its function of forming a processing platform or environment for services.

Two aspects need to be distinguished. First, there the (physical) aspect of deploying services to servers, which involves installation and configuration of programs and data for services on that server. The second aspect addresses the problem of making decisions about identifying locations for services using appropriate decision-making algorithm. We refer to the latter aspect in the following ignoring the (also hard) problem of service deployment and redeployment, which includes determining services' states.

Identifying a mapping (or making a service placement decision) is dependent on a set of policies and objectives to be achieved by a solution. It also depends on the "quality" of the algorithm and the "time" the algorithm needs to compute the decision. The classical tradeoff between quality and time appears to be fundamental in the domain of decentralized decision-making algorithyms.

A common objective is matching of service demand with resource supply (i.e. capacity of the hosting servers). Resource capacities should also be provided locally to demands avoiding cross-network traffic. This objective has been also adapted to function properly for algorithms which have only partial knowledge of the system state.

The requirements are dictated by the environment in which the algorithms are applied.

Requirements on the Algorithm Design

The scale and level of adaptability of such service networks dictate the following requirements on the algorithms for the resource placement problem.

Scalability and Decentralization. The design of an automated management system is closely related to the scale of the managed system and the rate of system changes. In an ideal case, current information about the state of the whole state can be collected in a centralized database, and as a consequence an optimal placement is made (if the problem is computationally tractable). However, with increasing scale and rate of system changes, this control solution becomes inappropriate, since the database is likely to become a bottleneck due to consistency updates. In addition, vulnerability to failures of a centralized solution results in low fault resilience.

Instead of a centralized solution, we consider decentralized algorithms which function under the assumption that each control element has only partial information about the global state of the system. While this leads to increased scalability, its drawbacks are non-optimal placements and higher complexity of the management infrastructure.

Adaptation. Large-scale peer-to-peer (P2P) systems such as file-sharing systems exhibit two properties which are cornerstones of automated resource management: adaptation and self-organization. We attempt capturing both properties in our algorithms by building on ideas from agent or robot coordination and P2P systems.

Adaptation is the ability to detect and react to changes in the state of the system, such as resource availability and service requirements. In a system comprising thousands of servers, changes such as server failure, overload, or resource revocation might occur every few seconds. Similarly, resource demand will fluctuate in short time intervals. These effects require adaptation of the system to new conditions on a permanent basis, ideally without human intervention. Such an automated service control system then transparently regulates service demands and supplies.

Self-Organization. We understand self-organization as the capability of adding and removing system parts without the need for manual reconfiguration or human intervention. This aspect is of particular interest since (non-automated) management of systems is an essential cost factor and a dominant source of error in IT environments.

Decentralization, Responsiveness and Placement Quality

A primary design goal of a placement algorithm is to let it find a "best" placement under the metric such as given in section "Placement Objectives and the Partial Objective Function (POF)". However, there are several obstacles to attain such a solution. One of them stems from the fact that the placement problems are known to be NP-complete. This requires heuristic approaches, but even such approaches have different running time, which influences the responsiveness of a system. responsiveness is understood as the time between detection of an abnormality, for instance a sudden peak demand, and the final computation of a decision how the situation can be dealt with.

The second obstacle is dictated by the requirement of the decentralization, which in most cases prevents the algorithm (or its parts) to know the current state of the whole system. Of course, partial knowledge leads in general to suboptimal placements and influences adversely the responsiveness.

In practice, decentralization and responsiveness of an algorithm must be traded against the quality of a solution. Figure 1 classifies four algorithms in regard to solution quality vs. responsiveness. Three time scales are considered: the "design" stage of an initial service placement, in longer periods reiterated as long-term adjustment process in the system; a mid-term period for periodic operational adjustments, and a shorter-term period for discharging sudden hot spots. One approach we pursued is a centralized heuristic algorithm based on integer programming.

This algorithm yields high-quality solutions but at a cost of longer running time and limited scalability, and is therefore not discussed here. The three algorithms described in this paper take different places in the trade-off space, and so the choice which one to deploy will depend on one of the management goals discussed above.

General objectives. The optimization goals for service placement might vary in general and so the presented algorithms are designed to be generic enough to support new objectives without fundamental changes. We focus on only few aspects to be achieved by control decisions. These are:

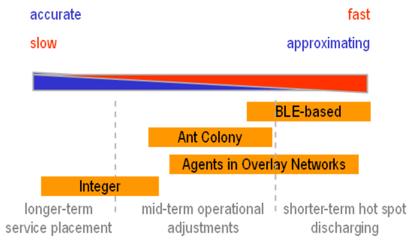

accurate
slow
fast
approximating
BLE-based
Ant Colony
Agents in Overlay Networks
Integer
longer-term service placement
mid-term operational adjustments
shorter-term hot spot discharging

Fig. 1. Comparison of algorithms regarding accuracy and responsiveness

1. Balancing the server load such that the utilization of each server is in a desired range.
2. Placing services in such a way that communication demand among them does not exceed the capacity of the links between the hosting server environments.
3. Minimizing the overall network traffic aiming to place services with high traffic close to each other on nearby servers (nearby in the sense of a low number of communication hops across nodes).

The Partial Objective Function. We want to be able to compare different placement options in a quantitative way. To this aim we introduce a partial objective function (POF) f_{POF}, which ranges between 0 and 1 and attains higher values for better placements (i.e. should by maximized).

The function is derived from a balanced sum of two characteristics. The first one, c_T, is the sum of traffic costs between the services on a pair of servers weighted by the distance of these servers. The second number, u_T, is the variance of the processing capacity usage among the servers. This leads to the POF computed by the formula:

$$f_{POF} = \frac{\beta}{\beta + (\alpha \cdot c_T + (1-\alpha) \cdot u_T)},$$

where α is the balancing factor between 0 and 1, and β a parameter described below. Higher values of α gives c_T more weight in the result, while lower values favors u_T.

In our setting, both a lower weighted traffic cost and a lower variance are better. This is reflected in the value of the POF, which has a higher “score” for smaller c_T or u_T. β must be chosen according to the maximum possible values of c_T and u_T in order to ensure a relatively uniform distributions of the values of the POF.

Our POF is evaluated for a set V of servers and a set S of services. It is important to note that such a set might not contain all services or all servers in the system. In case of services this is motivated by a fact that for larger systems we can frequently isolate groups of interdependent services (i.e. services communicating with each other). While it makes sense to consider all services in such a service group for a particular placement, we do not need to consider services outside the group.

The rationale for considering only few and not all servers is dictated by scalability issues. In large systems, it is simply impossible to take all servers into consideration. The algorithms described in the following select an appropriate subset of the servers from the system in a heuristic fashion. The subsets are then evaluated in the POF.

In the following, we give formal definitions for the characteristics c_T and u_T. We assume a fixed assignment of services in the set S to the servers in the set V.

For two servers v and v', we designate by $c_{v,v'}$ the estimated total traffic between all services placed on v and all services placed on v', measured in the number of exchanged IP packets. If $prox_{v,v'}$ is the network distance of servers v and v' (in terms of IP-hops), then the total weighted communication cost c_T is given by

$$c_T = \frac{1}{M} \sum_{v \in V} \sum_{v' \in V} prox_{v,v'} \cdot c_{v,v'},$$

where M is the total number of exchanged IP packets times the maximum distance between two servers in V. If a server v sends k packets to server v' and the distance between both servers is d (IP-hops), this message will contribute (kd)/M to c_T.

For a server v, let u_v be the fraction of its processing capacity used by all services placed on this server. We assume that u_v is a real number in [0, 1]. By rewriting the formula for a variance of a random variable, we obtain the variance u_T as:

$$u_T = \sum_{v \in V} u_v^2 - \frac{1}{|V|} \left(\sum_{v \in V} u_v \right)^2 .$$

Necessary conditions of an assignment. An assignment must fulfill certain necessary conditions; for example, we cannot assign a service to a server with insufficient processing capacity. By slightly abusing the notion of an "objective function", we can use f_{POF} to ensure that such requirements are fulfilled. Specifically, we set the value of the POF to 0, if any of the following conditions is violated:
Each service is placed at exactly one server.

- For each server v, the total processing demand of all services assigned to v is at most the processing capacity of v.
- For each server v, the total storage demand of all services assigned to v is at most the storage capacity of v.
- For each pair (v, v') of servers, the total network traffic between the services hosted on these servers should not exceed the link capacity between v and v'.
- The entries of a so-called affinity/repulsion matrix, if present, are respected; they indicate that a service must not or must be placed on a certain server.

Ant-Based Control Algorithm

Swarm Intelligence [3] is an approach where the collective behavior of simple agents with local interaction results in distributed problem solving without central control. The main application domains are optimization problems (such as Traveling Salesman Problem (TSP)) and telecommunication questions such as routing [30].

In a particular approach, the Ant Colony Optimization [6], the path taken by an ant on its way between objects (e.g. cities in TSP) represents a possible solution to the optimization problem. In our case, the objects would be both servers and services, and the alternating path would represent an assignment of services to servers.

However, this approach is centralized and not scalable for the following reasons:

1. The ant must "remember" the whole path it has taken; this information can become very large.
2. The ant must visit all objects on its tour. In a large and dynamic system, this is a serious drawback.
3. Finally, each solution (path) must be evaluated against others. This requires central knowledge.

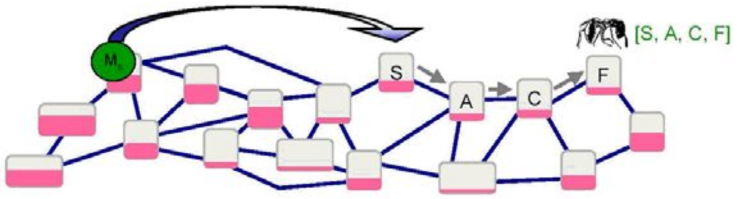


Fig. 2. In each step, the ant assigns one of the services from its service list

Overview

We describe another approach not common in the classical Ant Colony Optimization yet leading to a better scalability. First we give an informal overview of this algorithm. In our system, for each service s we instantiate a "demon" M_s called a service manager of s. If the service s is not yet placed or an overload condition has occurred, M_s creates multiple ants ("agents") and sends them out to the network. Each ant has a service list containing s and the services cooperating with s. For each such a service, it knows the current resource requirements; also, it knows the current communication requirements among the services in the service list.

The ant travels from one server to another choosing the servers along the path based on a probability computed locally. In each step, one of the services from the list is assigned to the current server, see Figure 2. The path created in this way represents a partial solution to the placement problem as found by this particular ant. When the ant has assigned all the services, it reports its path to the service manager M_s of s and terminates. The manager compares the reported paths using the POF, where the servers visited by this ant constitute the set V of the POF, and the set S is the service list of s. This assignment is compared with the current placement of those services. Finally, M_s decides of a possible rearrangement of the placement.

On each server, the ant evaluates the score of the server in respect to each service from its list. For each pair (service, server), this placement score expresses how well this server is suitable to host the service. It is computed also using the POF in the way described below. Furthermore, the ant causes the pheromone table of the current server to be updated. This table contains pheromone scores for certain pairs (service, server). Those are essentially weighted sums of placement scores of the ants that evaluated this particular (service, server)-pair. The table is used to help an ant to decide which server to visit next. The server managers of neighboring servers periodically exchange these tables, thus providing a mechanism to disseminate the local information across the system.

Ants, Service Managers and Server Managers

In our algorithm we have three entities that store and manipulate data:

- a service manager M_s of a service s,
- an ant representing s,
- a server manager (corresponding to a single server) which executes the ant code, and maintains and updates the pheromone table of its server.

The data held by a service manager comprises the service list of s, the number of spawned ants and the currently best assignment reported by an ant. A service manager also knows how to evaluate the POF and its value for the current placement of the services in the service list.

An ant is launched with the following data that are "static" during its lifetime: the service list together with the current demand profiles of each service in the list, and the communication demand profiles between those services. This information is necessary to compute the score via a POF. The dynamic data carried by an ant are the scores of the already assigned services from the service list, and data about already visited servers, including link capacities.

Table 1. An example pheromone table

serviceId	serverId	pheromone score	score age (sec)	# ants
apache-01	15.1.64.5	0.572	95	15
apache-01	15.1.64.7	0.356	120	9
oracle-02	15.1.64.1	0.012	62	12

Finally, a server manager holds the pheromone table of its server. The structure of the pheromone table is shown in Table 1. For each pair (serviceId, serverId) existing in this table, we record the known pheromone score, the age of this score and the number of ants which contributed to establish this score.

Functionality of the System Components

In this section we describe in detail the behavior of the entities introduced above.

Service managers. A service manager constantly watches the performance of "its" service and evaluates the current assignment by a POF. On two occasions it spawns ants starting a process described below:

- If the POF value is larger than some critical limit; this corresponds to the case of an occurrence of a "hot spot".

- If a certain period of time has passed since the last launch of the ants. The purpose of this step is to periodically "rebalance" the whole system towards an optimal utilization.

The process from the decision of launching ants until its termination includes the following steps:

1. Synthesize the ant data described in the section "Ants, Service Manager and Server Manager".
2. Place c_s copies of such an ant on the servers. The placement method and the value of c_s is described in the section "Initial Placement of Ants".
3. Collect the assignments and the corresponding scores sent by the ants which terminated.
4. Once all ants have finished (or a timeout has occurred), compare the reported assignments by the POF and choose the one with the best POF value.
5. If the service s has already been placed, compare the current POF of s and the cooperating services with the one found in Step 4. If the new assignment is better by a threshold t_s (representing the "penalty" for reassigning services to servers), continue with the next step; otherwise, terminate this epoch of ant launching.
6. If s is not placed, or the evaluation in Step 5. led to this step, reassign the services to servers in a following way.
    a. Contact all servers to be used in the new assignment of services and verify that their scores are still (approximately) valid. If this is not the case, start a new epoch of ant launching (i.e. begin from Step 1.)
    b. Contact the service managers of all cooperating services and let them stop any running ant-based evaluations.
    c. Contact the servers to be used in the new assignment and let them reserve the required resource capacities.
    d. Start installing and starting the services on their new locations.
    e. When step d. is finished, shut down services in the old placement.
    f. Finally, start the service managers of the newly installed services.

Ants. An ant created by a service manager M_s of a service s "travels" from one server manager to the next one (usually residing on an another physical entity). Technically, it is done by contacting the next server manager, transmitting the ant data to it and initiating executing the ant code for this ant instance. The choice of the next server manager is done in the way described below.

The ant has the following life cycle after it has arrived on a new server manager:

1. Evaluate for each service in the service list the score in regard to this server. This is done via the POF for this server as described in the section "Placement Scores and the Pheromone Trail".
2. Update the pheromone table of the current server by passing the computed scores to the server manager.
3. Choose the service with the highest computed score among the not yet assigned services and remember this assignment.

4. If all services from the internal list have been assigned, report the resulting assignment to the "original" service manager M_s, then terminate.
5. Otherwise, move to the next server manager and continue with 1.

Server managers. Both entities described above have essentially a fixed order of tasks to be executed. By way of contrast, a service manager acts in an asynchronous way, providing "services" to the other two entities. Its roles comprise following tasks:

1. It provides an environment where the ants are executed. Especially, it can asynchronously receive messages from other server managers that send the ant data. Once this data is received, it executes the locally stored code representing an ant.
2. It lets an ant update the pheromone table with the scores computed for the services in the service list.
3. It maintains the pheromone table by updating the age of the pheromone scores and pruning the table. The last step is necessary, because in the extreme case, the pheromone table could attain a size proportional to the number of servers multiplied by the number of services; this would seriously impede scalability. During the pruning, the oldest entries (except for those regarding the neighboring servers) are removed, until the desired table length is reached.
4. Finally, a server manager sends periodically its own pheromone table to the neighboring servers, keeping the information of the neighbors up to date.

The last function provides a mechanism for dissemination of the local knowledge throughout the system. This reduces the gap between a distributed system where each participant has only local knowledge, and a centralized system with the complete, global knowledge of the system. The size of the time interval between the updates and the size of a pheromone table controls the degree of the "global knowledge" in the system. An antagonistic trend is the rate of changes in the system and consequently the ageing rate of the pheromone. Also, albeit a high degree of this knowledge is very useful for choosing the next server in a correct way, attaining it costs a lot of resources, mostly network bandwidth and storage for the pheromone tables.

Placement Scores and the Pheromone Table

Recall when an ant reaches a new server manager, it computes placement scores for all services from its list in respect to the current server. For such a pair (service s, server v), this computation is done via the POF as follows. The current server v and servers already visited by an ant become the set V. Furthermore, s and all already assigned services from the ant's service list constitute the set S. Then the value of the POF is computed for the (partial) assignment of services to servers already chosen by this ant, together with the mapping of s to v. We assume that information about link capacities between the servers is buffered by the ant or can be obtained from the server manager, if necessary.

Let us describe now how the pheromone tables are updated. Assume that an ant has computed a fresh placement score r for the pair (service, server). If such a pair does not exist in this server manager's table, it is simply inserted with the pheromone score being equal to the placement score. Otherwise, the new value p' for this pair's pheromone score is computed from the current table entry p and the newly computed placement score r by the formula $p' = \gamma \cdot p + (1-\gamma) \cdot r$.

Here γ is a parameter between zero and one which determines the degree of inheriting previous pheromone score value. Note that the contribution of all other scores decreases geometrically with the number of iterations: if the very first ant which has visited the node has set the pheromone score to p, then after k new ants have reached the server, the contribution of this first ant to the current score of the pair will only be $\gamma^k p$.

We also want to consider an effect known from the ant colony systems in the nature: evaporation of the pheromone. Due to this effect, old and probably outdated information about affinities of services to servers will be removed with time, even if no new ants have arrived at this server. To this aim, a server manager scans through its pheromone table once every T minutes, and reduces the score p in the pheromone table according to the formula $p = \delta \cdot p$, where delta is an aging factor between 0 and 1 (usually close to 1). If the value of the pheromone score decreases below a certain limit, pairs are removed from the pheromone table in order to save storage resources.

Choosing the Next Server

Pheromone tables are the main decision factor for choosing the next server to be visited by an ant. Since those tables are exchanged by the neighboring servers and propagated through the system, an ant has a good chance to find a pair (service s, server v) in the pheromone table of the current server. Here v is a not too distant server and s is a still unassigned service from the service list of this ant. If multiple such pairs have been found, the server of a pair with the highest pheromone score is selected. However, if no such a pair exists, the ant chooses a set of servers from the pheromone table with 1. most recently updated pheromone scores, and with 2. highest pheromone scores. Then a random server from such a set is selected as the next host. This approach targets to identify servers with free computational resources.

As an alternative to each of the above cases, sometimes we send an ant to a randomly selected not-too-distant server. The decision for this step is taken with a (small) probability h. Such an addition of a "noise" is helpful to prevent the *blocking problem* and the *shortcut problem* [30]. The blocking problem occurs if a "popular" path found by many ants can no longer be taken, e.g. due to a server failure. The shortcut problem occurs in a situation where a new assignment of services to servers suddenly becomes possible, for example due to introduction of new servers to the system. In both cases the information stored in the pheromone tables might cause lack of adaptation of the ants to the new conditions. A small amount of noise forces the ants to exploit the alternative routes on a permanent basis.

Initial Placement of Ants

The initial placement of ants is intuitively an important factor for finding good service placements. In our case, the service manager M_s places the ants in the system according to the following schema.

First, it determines N_r "regions" where clusters of ants are placed. The centers of these regions are chosen randomly in the known system area in the way that the probability of choosing a center distant from the service manager is smaller than choosing a center close to M_s. To this aim, each service manager maintains a (partial) map of the resources in term of their network location. The resources are categorized by their IP-distance d to the server manager. When choosing a center of the region, in the fist step the service manager selects randomly a class of resources with a distance d to M_s. Then it decides to continue with this class with probability

$$\frac{1}{(1+d)^{\theta}},$$

otherwise it chooses again a random class until success; here θ is a parameter greater 1. If successful, a random resource as the center of a new region is chosen. The above formula yields high probability values for resource classes close to M_s (i.e. with small d), and rapidly decreasing probability values with growing distance d. According to the findings in [19], this approach ensures that very rare resources can be still discovered, but simultaneously supports clustering of services according to the location of their inception.

In each of the regions determined in this way, the service manager spawns N_a ants on the resources close to the center of the region. Here a similar approach to the one described above is taken, yet the distances of the created ants from the center of the region are kept smaller by means of increasing θ. Furthermore, ants "repel" themselves: if an ant is placed on a certain resource, then M_s will discard all servers within a distance D_r from this resource for further placements.

BLE-Based Control Algorithm

We adapt the concept of the Broadcast of Local Eligibility used for coordination of robots [32] for the placement of services. This concept can be used to create highly fault-tolerant and flexible frameworks for coordination of systems of agents. However, the originally proposed framework has a drawback of limited scalability. To overcome this problem, we use a hierarchical control structure discussed below.

Decision cycle in a cluster. We consider a cluster of servers with a distinguished server called cluster head. Each member of the cluster has the ability to broadcast a message to all other members of the cluster. This can be done either directly or via the cluster head. The placement of services in this cluster is periodically re-evaluated by arbitration between peer servers in so-called decision cycles. The time between two cycles is determined by the required responsiveness to fluctuations in server utilization and by the induced communication between cluster members.

In each decision cycle, the following actions take place:

1. Each server broadcasts the list of services it hosts with all new arrived services and simultaneously updates its list of all services in the cluster.
2. Each server evaluates its own suitability to host each service and sorts the list according to the computed score. The evaluation is done by using the POF from the section "Placement Objectives and the Partial Objective Function (POF)". In addition, a service already deployed on a server highly increases the score.

Each server broadcasts a list, ordered by scores, of those services the server can host simultaneously without exceeding its capacity.

3. When a server receives a score list from a peer, it compares this score with its own score for a service. As a consequence, each server knows whether it is the most eligible one for hosting a particular service.
4. The changes in the service placement are executed. Notice that each server knows already whether it has to install new or remove current services. In addition, the cluster head compares the initial list of services with those, which will be hosted at the end of this decision cycle. The remaining services are passed on to the next hierarchy level as explained below.

An important aspect is that the servers do not forget the list of services in the cluster after a decision cycle. In this way we provide fault-tolerance: if a server hosting certain services fails, other servers in the cluster will automatically install the failed services (or the cluster head adds them to the list of unassigned services).

Gossiping algorithms. Note that steps 1 and 3 require *all-to-all communication*, i.e. each server learns the information from all other servers. This may lead to a problem of the communication costs in terms of the number of messages and the time until all members of a cluster are informed. In infrastructures like Ethernet or wireless LAN a cost of a broadcast is comparable to sending a targeted message, which partially relieves the situation. This problem becomes more serious if members of a cluster are geographically distributed or communicate over a switched network.

These communication costs can be reduced using *gossiping algorithms* [13]. These deterministic and also randomized [18] algorithms achieve optimal bounds for the number of messages with a low number of communication rounds; for example, the information exchange can be completed in approximately $2\log_2 n$ steps in the deterministic case, and in roughly log n steps in the randomized case, where n is the number of servers in the cluster. The reader is referred to the literature for more detailed discussion.

Scalability by a cluster hierarchy. Obviously, the scalability of the above approach is limited by the size of the cluster, the communication capacity in the cluster and the processing capacity of the cluster head.

We propose a following hierarchical approach to extend the scalability. Basically, the cluster heads of the clusters at level k are treated as "normal" members of a cluster of level k+1. However, they compete only for those services, which could not be installed in their own cluster (see step 5. above). After a decision round in the cluster of level k+1, these pending services are possibly moved to another peer, which is a cluster head for a cluster of level k. (The cluster head evaluates the eligibility of the servers in its own cluster, not its own eligibility). In the cluster of level k, these services become part of the list of services to be installed and participate in the normal decision cycles.

The cluster size is essential for the balance between the responsiveness of the system and flexibility. Identifying a correct hierarchical structure can be done similarly to clustering algorithms used in sensor networks [8].

Agents in Overlay Networks

In this section we describe an approach that combines the advantages of agent technology techniques with the fault-tolerant properties of peer-to-peer (P2P) networks.

Service groups and agents. Services frequently build clusters of interdependent entities, which do not rely on further services outside the cluster. Such a service group, if not too large, can be treated as one (albeit not atomic) entity in the process of the optimization. Therefore we assign to such a service group N_a instances of group agents. Each group agent has the task to walk around in the network and evaluate the current server and its neighborhood in regard to placement of the services in the service group; however, one agent stays on one of the servers which host members of the service group, and evaluates only the current placement.

The evaluation of potential new placements is initiated by retrieving the capacity parameters and utilization data of the current server and its neighboring servers by means of a P2P-network described below. This data is then a subject to evaluation by the Partial Objective Function from the section "Placement Objectives and the Partial Objective Function (POF)". Periodically, the group agents belonging to the same service group exchange their best scores. If the score of one of them is better than the real placement (also taking into account a penalty for moving services), this group agent initiates a rearrangement of the placement.

A further assignment of a group agent is to provide the fault-tolerance to the optimization infrastructure: it is done by constantly watching all other N_a–1 group agents for being alive; the special group agent staying close to actually deployed services also watches the health of the services. If one of the group agents fails, it is immediately re-instantiated by other agents. Also, if one of the services turns out to have failed, an appropriate recovery action is initiated.

It is important to note the difference to the Ant Colony Optimization Algorithm presented in the section "Ant-based Control Algorithm". While both ants and agents use a notion of a service group and carry data of services in such a group, agents have a different evaluation algorithm compared to ants. While an ant assigns a service to a server in each step, an agent evaluates a possible assignment of all services to the current server and its neighbors in such a step. Furthermore, agents have more "intelligence" and do not die as opposed to ants. On the other hand, ants use the pheromone trails to learn the best assignments.

P2P-based overlay networks. Since the evaluation of a new agent placement incurs a lot of effort, the next jump of an agent must be chosen carefully. To this aim agents are guided by information from an overlay network that provides capacity-related attributes of servers. In the overlay network described in [2] servers are connected in a P2P-manner to achieve fault-tolerance and self-organizing properties (i.e. servers may join and leave without a reconfiguration exercise). The functionality of the network allows range queries of attributes; in our case we are mostly interested in server processing capacity, server storage capacity and the density values of these attributes. The density of an attribute is the averaged attribute value from a group of servers whose center is the server which "labels" this density value; thus, a density value is an indicator of the attribute (capacity) in the surrounding of a server. Density values are periodically computed on each server by updates received from the surrounding resources. When deciding about the next server to be visited, an agent first collects the current utilization data from its service group. This demand value determines the range for which the density values are queried. The overlay network responds with a list of servers fulfilling the criteria. An agent sorts them according to their distance, and chooses randomly the next server to move on, similarly as described in the section “Choosing the Next Server”. Once arrived on the new server, it queries directly the surrounding servers retrieving their individual attribute values. If ranges of values are necessary, the overlay network query capability can be used. This data is then used for the evaluation of the POF.

Conclusion

The algorithms presented in this paper provide means for distributed placement of services on servers in Autonomous Service Networks, for example global IT environments, large data centers, or Grids. The assumptions of large scale, lack of global knowledge and constantly changing operating conditions result in the design of distributed, heuristic algorithms. The algorithms build on techniques from agent coordination and P2P systems, trying to capture their properties of adaptability and self-organization.

A qualitative comparison and classification of the algorithms is provided in Table 2 (a comparison to a centralized integer programming approach not discussed in this paper is also provided). Obviously none of the approaches covers all desirable requirements; in particular, the tradeoff between responsiveness and solution quality becomes apparent.

Future work in this are will include refinement of the algorithms (including design alternatives) and integration in a Grid management infrastructure.

Table 2. Comparison of the algorithms

Approach	Scalability	Accuracy	responsiveness	Self-organization	Fault resilience	Extensi-bility	Simplicity
Integer	-	+	-	-	-	+	+
Ants	+	-	-	+	+	-	-
BLE	+	-	+	+	+/-	-	+
Ovl. Agents	+	-	+/-	+	+	+	-

References

1. Andrzejak A, Graupner S, Kotov V, Trinks H. Self-Organizing Control in Planetary-Scale Computing. IEEE International Symposium on Cluster Computing and the Grid (CCGrid), May 21-24, 2002, Berlin.
2. Andrzejak A, Xu Z. Scalable, Efficient Range Queries for Grid Information Services. Second IEEE International Conference on Peer-to-Peer Computing (P2P2002), Linköping, Sweden, 5-7 September 2002.
3. The Anthill Project. University of Bologna. http://www.cs.unibo.it/projects/anthill/.

4. Bonabeau E, Théraulaz G. Swarm Smarts, Scientific American 2000;72-9.
5. Brewer EA, Katz RH, Amir E, Balakrishnan H, Chawathe Y, et al. A Network Architecture for Heterogeneous Mobile Computing. IEEE Personal Communications Magazine, Oct. 1998.
6. Dorigo M, Maniezzo V, Colorni A. The Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man, and Cybernetics, Part B, 1969;26(1):29-41.
7. El-Rewini H, Lewis TG, Ali HH. Task scheduling in parallel and distributed systems. PTR Prentice Hall; 1994.
8. Estrin D, Govindan R, Heidemann J, Kumar S. Next century challenges: Scalable coordination in sensor networks. Proc MOBICOM August 1999; 263-270.
9. Foster I, Kesselman C. The Grid: Blueprint for a New Computing Infrastructure. 2nd ed. Morgan Kaufman; 2003.
10. Foster I, Kesselman C, Nick JM, Tuecke S. The Physiology of the Grid – An Open Grid Services Architecture for Distributed Systems Integration. May 2002, http://www.globus.org/research/papers/ogsa.pdf.
11. The Globus Toolkit, http://www.globus.org/toolkit.
12. Graupner S, Kotov V, Andrzejak A, Trinks H. Service-Centric Organization of Globally Distributed Computing. IEEE Internet Computing, special issue on "Grid Computing"; July/August 2003; 36-43.
13. Hedetniemi ST, Hedetniemi SM, Liestman AL. A survey of broadcasting and gossiping in communication networks. Networks 1988; 18:319-349.
14. Hewlett-Packard company. Utility Data Center. http://www.hp.com/go/hpudc, http://www.hp.com/go/always-on.
15. IBM. http://www.ibm.com/grid.
16. IBM, University of Berkeley. Oceano Project. http:/ /www.research.ibm.com/ oceanoproject.
17. IBM. Autonomic Computing Manifesto. http://www.research.ibm.com/autonomic/ manifesto
18. Kermarrec AM, Massoulie L, Ganesh AJ. Reliable Probabilistic Communication in Large-Scale Information Dissemination Systems. Microsoft Research Technical Report MMSR-TR-2000-105, October 2000.
19. Kleinberg J. The Small-World Phenomenon: An Algorithmic Perspective. Cornell Technical Report 99-1776, October 1999.
20. Kotov V: On Virtual Data Centers and Their Operating Environments, HP Labs Technical Report, HPL-2001-44, March 2001.
21. Krauter K, Buyya R, Maheswaran M. A Taxonomy and Survey of Grid resource Management Systems. Software-Practice and Experience 2002; 32(2):135-164.
22. Kubiatowicz J, Bindel D, Chen Y, Czerwinski S, Eaton P, et al. OceanStore: An Architecture for Global-Scale Persistent Storage. ASPLOS '00.
23. Luo QY, Hendry PG, Buchanan JT. Comparison of different approaches for solving distributed constraint satisfaction problems. Research Report RR-93-74, Department of Computer Science, University of Strathclyde, Glasgow, 1993.
24. Marcus E, Stern H. Blueprints for High Availability: Designing Resilent Distributed Systems. John Wiley & Sons, 2000.
25. Montresor A, Meling H, Babaoğlu Ö. Messor: Load-Balancing through a Swarm of Autonomous Agents. Proc 1st International Workshop on Agents and Peer-to-Peer Computing, Bologna, Italy, July 2002.
26. Platform Inc., http://www.platform.com.

27. asamy S, Francis P, Handley M, Karp R, Shenker S. A Scalable Content-Addressable Network. SIGCOMM 2001.
28. Rolia, J, Singhal S, Friedrich R. Adaptive Data Centers. Proc SSGRR 2000 Computer and eBusiness Conference, L'Aquila, Italy, August 2000.
29. Royer EM, Toh C-K. A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks. IEEE Personal Communications Magazine April 1999.
30. Schoonderwoerd R, Holland O, Bruten J, Rothkrantz L. Ants for Load Balancing in Telecommunications Networks. Adaptive Behavior 1996;2:169-207.
31. Sun Microsystems. The Sun Grid Engine. http://wwws.sun.com/gridware.
32. Werger BB, Matarić M. From Insect to Internet: Situated Control for Networked Robot Teams. To appear in Annals of Mathematics and Artificial Intelligence.

Author Index