SpringerLink

Book

Mobile Agents for Telecommunication Applications: Second International Workshop, MATA 2000, Paris, France, September 2000. Proceedings

Book Series	Lecture Notes in Computer Science
Publisher	Springer Berlin / Heidelberg
ISSN	0302-9743
Subject	Computer Science
Volume	Volume 1931/2000
Copyright	2000
Online Date	Thursday, July 31, 2003

21 Chapters

First | **1-10** | 11-20 | 21 | Next

Access to all content ■ Access to some content □ Access to no content

21 Chapters First | **1-10** | 11-20 | 21 | Next

SpringerLink

Book

Mobile Agents for Telecommunication Applications: Second International Workshop, MATA 2000, Paris, France, September 2000. Proceedings

Book Series	Lecture Notes in Computer Science
Publisher	Springer Berlin / Heidelberg
ISSN	0302-9743
Subject	Computer Science
Volume	Volume 1931/2000
Copyright	2000
Online Date	Thursday, July 31, 2003

21 Chapters

First | 1-10 | **11-20** | 21 | Next

Access to all content Access to some content Access to no content

SpringerLink

Book

Mobile Agents for Telecommunication Applications: Second International Workshop, MATA 2000, Paris, France, September 2000. Proceedings

Book Series	Lecture Notes in Computer Science
Publisher	Springer Berlin / Heidelberg
ISSN	0302-9743
Subject	Computer Science
Volume	Volume 1931/2000
Copyright	2000
Online Date	Thursday, July 31, 2003

21 Chapters First | 1-10 | 11-20 | **21** | Next

Access to all content Access to some content Access to no content

21 Chapters First | 1-10 | 11-20 | **21** | Next

Lecture Notes in Computer Science 1931
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Eric Horlait (Ed.)

Mobile Agents for Telecommunication Applications

Second International Workshop, MATA 2000
Paris, France, September 18-20, 2000
Proceedings

Springer

Volume Editor

Eric Horlait
Université Pierre et Marie Curie, Labo CNRS - LIP6
4, Place Jussieu, 75252 Paris Cedex 05, France
E-mail: Eric.Horlait@lip6.fr

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Mobile agents for telecommunication applications : second
international workshop ; proceedings / MATA 2000, Paris, France,
September 18 - 20, 2000. Eric Horlait (ed.). - Berlin ; Heidelberg ;
New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ;
Singapore ; Tokyo : Springer, 2000
(Lecture notes in computer science ; 1931)
ISBN 3-540-41069-4

CR Subject Classification (1998): C.2, I.2.11, H.4.3, H.5, J.1, J.2

ISSN 0302-9743
ISBN 3-540-41069-4 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

Printed in Germany

Typesetting: Camera-ready by author
Printed on acid-free paper SPIN 10781161 06/3142 5 4 3 2 1 0

Preface

Mobile agents refer to self-contained and identifiable computer programs that can move within the network and can act on behalf of the user or another entity. Most of the current research work on the mobile agent paradigm has two general goals: reduction of network traffic and asynchronous interaction. These two goals stem directly from the desire to reduce information overload and to efficiently use network resources.

There are certainly many motivations for the use of a mobile agent paradigm; however, intelligent information retrieval, network and mobility management, and network services are currently the three most cited application targets for a mobile agent system. The aim of the workshop is to provide a unique opportunity for researchers, software and application developers, and computer network technologists to discuss new developments in the mobile agent technology and applications.

After last year's very successful workshop in Ottawa, Canada (110 attendees), this year's workshop will focus on mobile agent issues across the areas of network management, mobile applications, nomadic computing, e-commerce, ad-hoc networks and applications, feature interactions, Internet applications, QoS management, policybased management, interactive multimedia, and computer–telephony integration.

September 2000 Eric Horlait

Scientific Program Committee

Mihai Barbuceanu, University of Toronto, Canada
Raouf Boutaba, Waterloo University , Canada
Jean-Pierre Briot, LIP6, France
Jean-Pierre Courtiat, LAAS, France
Jaime Delgado, Universitat Pompeu Fabra, Spain
Petre Dini, AT&T, USA
Andrzej Duda, LSR-IMAG, France
Benjamin Falchuk, Telcordia Technologies, USA
Nelson Fonseca, Unicamp, Brazil
Damianos Gavalas, University of Essex, United Kingdom
Valerie Gay, UTS, Australia
Marie-Pierre Gervais, LIP6, France
Roch Glitho, Ericsson, Sweden
Clifford Grossner, Nortel Networks, Canada
Eric Horlait, LIP6, France
Roger Impey, National Research Council, Canada
Bijan Jabbari, George Mason University, USA
Ahmed Karmouch, University of Ottawa, Canada
Ramiro Liscano, National Research Council, Canada
Thomas Magedanz, IKV++ GmbH, Germany
John Meech, Network Computing Group, IIT, Canada
Bernie Pagurek, Carleton University, Canada
Michel Riguidel, Thomson, France
Michel Soto, LIP6, France

MATA Workshop Steering Committee

Eric Horlait, LIP6, France
Roger Impey, National Research Council, Canada
Ahmed Karmouch, University of Ottawa, Canada, Chair
Thomas Magedanz, IKV++ GmbH, Germany

Table of Contents

Ad-hoc Networks and Applications

Network Management

Architecture and Methodologies – 1

Active Networks

Agent-Based Applications

Architecture and Methodologies – 2

Author Index

Topology Discovery in ad hoc Wireless Networks Using Mobile Agents

Romit RoyChoudhuri[1], S. Bandyopadhyay[2], Krishna Paul[3]

[1] Department of Computer Sc and Engg
Haldia Institute of Technology
Haldia, West Bengal, India

[2] PricewaterhouseCoopers, Saltlake Technology Center
Sector V, Calcutta 700 091, India
{somprakash.bandyopadhyay@in.pwcglobal.com}

[3] Cognizant Technology Solutions, Sector V, Saltlake
Calcutta 700 091 India
{PKrishna2@cal.cts-corp.com}

Abstract. Extensive research on mobile agents has been rife with the growing interests in network computing. In this paper, we have discussed a mobile multi-agent-based framework to address the aspect of topology discovery in ad hoc wireless network environment. In other words, we have designed a multi-agent based protocol to make the nodes in the network *topology aware*. Our primary aim is to collect all topology-related information from each node in the network and distribute them periodically (as updates) to other nodes through mobile agents. The notion of *stigmergic communication* has been used through the implementation of a shared information cache in each node. Moreover, we have defined a concept of *link stability* and *information aging* based on which a predictive algorithm running on each node can predict the current network topology based on the current network information stored at that node. We have demonstrated through performance evaluation of a simulated system that the use of mobile multi-agent framework would be able to make each node in the network *topology aware*, without consuming large portion of network capacity. As a direct outcome of infiltrating topology information into the nodes, the foundations for designing efficient routing scheme, distributed network management and implementing communication awareness get automatically laid.

1. Introduction

A mobile agent is a program that can move through a network under its own control, capable of navigating through the underlying network and performing various tasks at each node independently [1]. Mobile agents are an effective paradigm for distributed applications, and are particularly attractive in a dynamic network environment involving partially connected computing elements [2]. In this paper, we propose to use a mobile multi-agent-based framework to address the

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 1-15, 2000.

aspect of topology discovery in a highly dynamic ad hoc wireless network environment [3,4,5,6]. In other words, we have designed a multi-agent based protocol to make the nodes in the network **topology aware**. As a direct outcome of infiltrating topology information into each node, the foundations for designing efficient routing scheme, distributed network management and implementing communication awareness [7] get automatically laid.

As an example, the dynamics of wireless ad hoc networks as a consequence of mobility and disconnection of mobile hosts pose a number of problems in designing proper routing schemes for effective communication between any source and destination[5]. The conventional proactive routing protocols that require to know the topology of the entire network is not suitable in such a highly dynamic environment, since the topology update information needs to be propagated frequently throughout the network. On the other hand, a demand-based, reactive route discovery procedure generates large volume of control traffic and the actual data transmission is delayed until the route is determined. Because of this long delay and excessive control traffic, pure reactive routing protocols may not be applicable to real-time communication. At the same time, pure proactive schemes are likewise not appropriate for the ad-hoc network environment, as they continuously use a large portion of the network capacity to keep the routing information current [4]. In this work, we have demonstrated through performance evaluation of a simulated system that the use of mobile multi-agent framework would be able to make each node in the network topology aware without consuming large portion of network capacity. This would eventually help us to implement a proactive routing protocol without much overhead.

Mobile agents or messengers that hop around in the network are a novel solution to the problem of topology discovery. The agents hop from node to node, collect information from these nodes, meet other agents in their journey, interact with both to collect updates of parts of the network that they have not visited or have visited a long time back, and gift these collected data sets to newly visited nodes and agents. A node therefore receives updated information about the network from the agents visiting them at short regular intervals. Initially when the network commences, all the nodes are just aware of their own neighbors and are in a *don't-know-state* regarding the other nodes in the system. However with agent navigation beginning, the nodes slowly get information about the other nodes and their neighbors.

For example, let us assume that an agent migrates at every K time tick between nodes. At time t_0, each of the nodes has only information about their immediate neighbors. At time = t_0 + K, an agent jumps to a node with the information that it has about its previous host node. Thus the current host node now has data about a new neighboring node and its neighbors (since the agent has carried this information to it). In the next K time tick, a node gets information regarding two more nodes from another agent. It is to be noted that by controlling K, it is possible to control the agent traffic in the network. Moreover, the agent would always migrate from a node to only one of its neighbor after K time-tick. So, the network would never get flooded with propagation of agents.

2. Related Work

Currently, there is a growing interest in using mobile agents as part of the solution to implement more flexible and decentralized network architecture [1, 8]. Most research examples of the mobile agent paradigm as reported in the current literatures have two general goals: reduction of network traffic and asynchronous interaction. Some authors have suggested that agents can be used to implement network management [9, 10] and to deliver network services [11].

Intensive research on the "Insect-like Systems" has been done over the last few years. The mobile agent systems have been popularly simulated in close resemblance to an ant colony [12]. Of particular interest is a technique for indirect inter-agent communication, called stigmergy, in which agents leave behind the information in the cache of the node that they have visited. Stigmergy serves as a robust mechanism for information sharing. Worker ants that leave pheromone trails when they venture outside their nest are using stigmergic communication. This notion has been used in [13,14]. Mobile agents are on the use for multifarious purposes ranging from adaptive routing [13], distributing topology information [14], offline message transfer[15] and distributed information management [16].

In this paper, our primary aim is to collect all topology-related information from each node in ad hoc wireless network and distribute them periodically (as updates) to other nodes through mobile agents. The notion of stigmergic communication has been used through the implementation of a shared information cache in each node. The basic idea of using agents for topology discovery has been explored in MIT Media Lab [14] earlier with certain limitations : first, node mobility and its effect on system performance has not been quantified. Second, the information convergence (convergence of the difference between actual topology information and the topology information as perceived by a node at any point of time) and its relationship with number of agents and agent migration frequency has not been clearly defined. Third, the navigation strategies used do not ensure a balanced distribution of recent topology information among all the nodes. We have tried to overcome these difficulties. Moreover, we have defined a concept of link stability and information aging based on which a predictive algorithm running on each node can predict the current network topology based on the current network information stored at that node.

3. Description of Relevant Terms

3.1 Link-Affinity

In a wireless environment, each node n has a wireless transmitter range. We define the neighbors of n as the set of nodes within the transmission range R of n. It is assumed that when node n transmits a packet, it is broadcast to all of its neighbors. However, in the wireless environment, the strength of connections to all the members of the neighbor set with respect to any node n are not uniform. For example, a node m in the periphery of the transmission range of n is weakly connected to n compared to a node u which is closer to n. Thus, the chance of m

going out of the transmission range of n due to an outward mobility of either m or n is more than that of u.

Link-Affinity α_{nm}, associated with a link between two nodes n and m, is a prediction about the span of life of that link in a particular context [7]. For simplicity, we assume α_{nm} to be equal to α_{mn} and the transmission range R for all the nodes are equal. To find out the affinity α_{nm}, node n sends a periodic beacon and node m samples the strength of signals received from node n periodically. Since the signal strength perceived by m is a function of R and the current distance r between n and m, we can predict the current distance r at time t between n and m. If M is the average velocity of the nodes, the worst-case average affinity α_{nm} at time t is (R - r)/M, assuming that at time t, the node m has started moving outwards with an average velocity M. If m and n are not neighbors of each other, $\alpha_{nm} = 0$.

3.2 Recency

A major aspect underlying the infiltration of topology information into mobile nodes is that the information carried must be recognized with a degree of correctness. Since the agent navigation is asynchronous and there is an obvious time gap between the procurement of information by an agent from one node and its delivery by the same agent to another node, it becomes imperative to introduce a concept of *recency of information.* For example, let us assume two agents A_1 and A_2 arrive at node n, both of them carrying information about node m which is multi-hop away from node n. In order to update the topology information at node n about node m, there has to be a mechanism to find out who carries the most recent information about node m : agent A_1 or agent A_2 ?

To implement that, every node in the network has a counter that is initialized to 0. When an agent leaves a node after completing all its tasks at the node, it increments that counter by one. We term this counter as *recency token*. An agent while leaving a node (after completion of all the necessary information exchange and calculations) stores the new value of the incremented recency token against the node's ID within its data structures and continues navigating. Thus at any point of time, the magnitude of the recency token of any node represents the number of times that node was visited by agents since the commencement of the network. This also implies that if two agents have a set of data concerning the same node, say node x, then the agent carrying the higher recency token value of node x has more current information about it. The far-reaching advantages derived from this scheme would be pointed out in further illustrations where the recency token is extensively used.

3.3 Time-to-Migrate (TtM)

An agent visiting a node is not allowed to migrate immediately to another node. In other words, an agent will be forced to stay in a node for a pre-specified period of time, termed as *time-to-migrate (TtM)*, before migrating to another node. By controlling TtM, the network congestion due to agent traffic can be controlled. For example, if TtM = 100 msec., for a single-agent system, it implies that the wireless

medium will see one agent in every 100 msec. In our simulation, it has been assumed that an agent would take 1 msec. to physically migrate from one node to another. So, in this example, the wireless medium would be free from agent traffic 99% of the time.

3.4 Average Connectivity Convergence

We have developed a metric *average connectivity convergence* to quantify the deviation of actual network topology with the network topology perceived by individual nodes at any instant of time.

Let l^a_{nm} be the link status (0 for disconnectivity and 1 for connectivity) between node n and m as perceived by node a at any instant of time and l_{nm} is the actual link status between node m and n at that instant of time. Information about link status l^a_{mn} is said to converge at node a iff $l^a_{mn} = l_{mn}$. Thus, connectivity convergence of link between n and m at node a, $\gamma^a_{nm} = 1$, if $l^a_{nm} = l_{nm}$ and 0 otherwise. *Connectivity convergence* of node a for all links in the network, γ^a is defined as: $\gamma^a = (\Sigma_{\text{for all node-pairs i-j}} (\gamma^a_{ij}))$ / total number of node-pairs

At some instant of time, if $\gamma^a = 1.0$, it implies that the topology information at node a is exactly the same as the actual network topology at that instant of time. As another example, in a 10-node network, there are 45 node-pairs and 45 possible link-status. If, at any node a, 44 link-status matches at any instant of time with the actual link-status, then $\gamma^a = 44/45 = 0.98$.

The *average connectivity convergence*, $\gamma_{avg} = (\Sigma_{\text{for all nodes k}} (\gamma^k))$ / number of nodes

3.5 Average Link-affinity Convergence

Average connectivity convergence quantifies the deviation of actual network topology with the network topology perceived by individual nodes in discrete manner (where link status is 0 for disconnectivity and 1 for connectivity). However, if we can quantify link status based on link-affinity, the quantification could be more appropriate in formulating a metric which would help us to evaluate the difference between the actual network topology and the network topology as perceived by individual nodes in a continuous scale.

Let α^a_{nm} be the affinity between node n and m as perceived by node a at any instant of time and α_{nm} is the actual affinity between node m and n at that instant of time. Information about link status α^a_{mn} is said to converge at node a iff $\alpha^a_{mn} <= \alpha_{mn}$. As indicated earlier, affinity is a worst-case prediction about the span of life of a link. So, if the affinity of a link between n and m as perceived by a node a is less than actual affinity between n and m, we will accept the perception of node a about the link-affinity between n and m. However, if $\alpha^a_{mn} > \alpha_{mn}$, we will consider this as an overestimate by node a about the link affinity between n and m and we reject the perception.

Thus, *link-affinity convergence* of link between n and m at node a, $\lambda^a_{nm} = 1$, if $\alpha^a_{mn} <= \alpha_{mn}$ and 0 otherwise.

Link-affinity convergence of node a, λ^a, for all links in the network, is defined as: $\lambda^a = (\Sigma_{\text{for all node-pairs i-j}} (\lambda^a_{ij}))$ / total number of node-pairs At some instant of time, if $\lambda^a = 1.0$, it implies that the topology information at node a is 100% acceptable, so far as affinity-based prediction mechanism is concerned.

The *average link-affinity convergence*, $\lambda_{avg} = (\Sigma_{\text{for all nodes k}} (\lambda^k))$ / number of nodes

4. Issues in Implementing Agent Paradigm

4.1 Single vs. Multiple Agents

The topology traversing could well be performed using a single agent. However this strategy fails to perform well in conditions of low transmission range where clusters gets formed due to groups of nodes, moving to some spatially remote portion of the bounded region. Quite obviously, since the agent is going to be in only one of the clusters, the other clusters have no agents at all in them although the members belonging to those clusters may be connected to quite a number of other nodes. Thus, the deprived clusters can no more get topology maps and might even get misled by the old, and thus incorrect, information they might be having regarding the connectivity of the network.

Although we have not addressed the issue of agent loss (during transit) in this paper, it can be well understood that the single agent system would also suffer in the event of the agent getting lost during transit. This consequently means that the system no longer has an agent and the nodes are not even aware of it, since each would indefinitely wait for an agent to come to it.

The above mentioned issues cause no serious concern in the case of a multi-agent system. There are two factors to be considered in multi-agent system :

The queuing factor: With the increase in number of agents, less and less number of nodes is free of agents at any instant of time. In other words, the average number of agents in the nodes' agent queues increase. Since more and more agents are held in agent queues, the effective number of agents in the system decreases. This decrease in the effective number of agents compensates for the advantage of having more agents in the system which otherwise means that the convergence could be better.

The issue of common intentions in multi-agent systems: As the number of agents increase, the information that each of them get after the exchanges become identical and thus the corresponding decisions that they take regarding migration gradually becomes similar [14]. As a result agents crowd together at certain parts of the topology and the very essence of homogeneous topology exploration gets lost. Thus, this inhibits the prospective gain in increasing agent population and the increase in agent population also increases the traffic/congestion in the system.

4.2 Choosing the Optimal Agent Population

It might be of interest to observe that average connectivity convergence does not necessarily improve with the increase in number of agents in the system. It is this observation in our work that served as the motivation to analyze the dependence of

performance on agent population and agent TtM. Thus we have determined the optimal number of agents in a system of N nodes to obtain the best optimal service.

Fig.1 show the analysis curves of average connectivity convergence against varying number of agents for different mobility and TtM = 100 ms.. It can be seen from the graph that the performance of the agency improves with greater number of agents up to a certain point after which the performance is not discernable. Increase in agent population in the system clearly indicates the increase in congestion in the system. Thus, to keep congestion under control, we rationally choose that agent population for which the agent traffic is tolerable and the requirements are well met. We have chosen the critical agent population to be *half the number of nodes* in the system.

4.3 Spawning of Agents

Each node at its start-up process generates a random number. We propose that, if the number is even, then the node spawns an agent in the network. This indicates that the network commences with an agent population which is roughly half the number of nodes in the network.

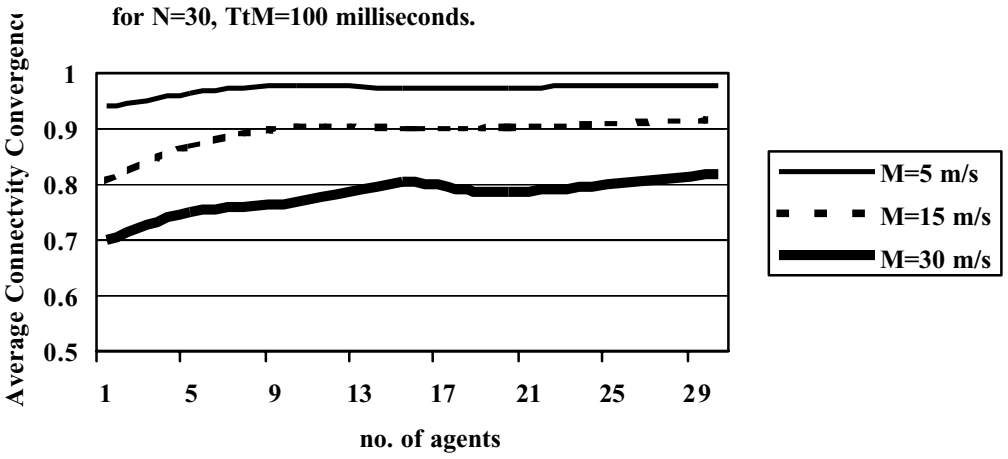


Fig.1. The variation of connectivity convergence against number of agents in the system, analyzed for different mobility values with TtM =100 ms

5. Topology Discovery Using Agents : Basic Mechanism

5.1 Navigation Algorithm for Optimal Throughput and Quick Convergence

The navigation algorithm must ensure that all member nodes of the network update themselves uniformly, irrespective of their position and state in the network. We

have designed an algorithm, called *least-visited-neighbor-first* algorithm, and implemented it in our simulation model to estimate its effectiveness.

In this algorithm the agents select destination nodes that has the least recency_token value, implying that the node has been visited less frequently. An agent, residing in node n at any instant of time (termed as current host node of the agent), does the following:

1. Update information cache of node n with the information available in its information cache.
2. Selects all the nodes that are neighbors of n.
3. Evaluates the minimum value of the recency token of these selected neighbors from the information cache. This is the least visited neighbor, as perceived by the agent's host node n at that instant of time.
4. If this neighbor of n has not been visited in the last 3 visits by other agents from node n, select this neighbor as next destination. This history information is stored in the nodes. Else, select the second least-visited neighbors, and so on. This will ensure that multiple agents from same host node do not choose the same destination consecutively.
5. After choosing the right destination, it updates its next_detination node id with the destination node's id, changes the history table of the host node n with this newly selected node id.
6. Increments the host node's recency token value and stores this value against the host node's id within its own information cache.
7. The agent resumes navigation.

5.2 Handling the Event of Agent Oscillation between Nodes

Let us consider a case where a node gets isolated from other nodes in the network and as a result does not receive agent visits for a long time. Now let us assume that it gets connected to the network once again after some time. Obviously, the recency token value of this node would be much less than the values of the others, which have continuously received agents at regular intervals. Now this isolated node on getting connected to the network would obviously have a rush of agents towards it. The agents would get serviced, go to some next destination and again come back to this newly connected node until the recency value of the newly connected node is no longer the least among all the recency tokens of the other nodes in the network. This means that if a node gets isolated and then rejoins, the agents would oscillate over it and its neighbors until its recency value surpasses some other neighbor's recency value. Oscillation might cause other nodes in the network to starve of agent visits and is unnecessary from the perspective of percolating current information.

In order to eliminate this oscillation, we have incorporated the following strategy. If a node finds that it does not receive agent visits for longer than a specific span of time, it resets its recency token value to zero. An agent that visits a node, finding that the node has a recency_token value of zero recognizes that the node performed a reset. The agent performs the standard node-agent information interchange and then assigns the average of all its recency token values to this node's recency token. This prevents the oscillation of the same agent. However,

other agents may independently come to this node since they might still have a recency value for this node as the least value. This is desirable in order to percolate a new node's information fast. Greater agent visits (i.e different agents visiting it) would catalyze this quick infiltration of information.

5.3 Information Exchange in Node-Agent and Agent-Agent Interaction

Infiltration of partial network information into the nodes is an asynchronous process, as the agents visit the nodes asynchronously. Thus it becomes necessary to develop strategies for information exchange (i.e. to accept only that information which is more recent than what the node / agent already possesses). It is a two-step process.

In step 1, the recency tokens of all the nodes stored in the information cache of the current host node and the recency tokens of all the nodes stored in the information cache of the agent is compared. If the recency token of any node, say X, in the host node's information cache happens to be less than that in the agent's information cache, then it is obvious that the agent is carrying more recent information about node X. So, the entire information about node X in the host node's information cache is overwritten by that in the information cache of the agent. This step is performed asynchronously for all the agents as they arrive at that host node. This step helps the node to acquire all the recent information that it can gather from the agents.

When an agent is ready to migrate (i.e. after a waiting time defined earlier as TtM), the step 2 is performed. In step 2, the agent copies the entire information cache of the host node on their own information cache. This contains the most recent information since the data set contains a combination of all the recent information that could be collected from the visiting agents and those that were already present in the node. With these updated values, the agents select their destination on the basis of navigation algorithm. The snap-shot in fugure 2 illustrates the result of the processes described in this section.

6. Information Aging: a Predictive Method for Topology Discovery

The foremost characteristic of a dynamic environment is that information is never absolute. Data sets gathered by nodes from agents on connectivity and link affinities are constantly changing. This might prove to be detrimental if not handled with care. For example, assume that a node receives information regarding the affinity of link xy at time t_0. Now if it wants to start a data transfer session after time t_1, it just might happen that the link ceases to exist within that time. However, the node remains oblivious of this until another agent brings to it this information about the link disconnection. To avoid such events in a highly dynamic scenario, agent TtM might have to be decreased which will eventually increase agent-traffic in the network.

We have defined a concept of *information aging* on link-affinity based on which a predictive algorithm running on each node can predict the current network

topology based on the current network information stored at that node. We apply this predictive mechanism to predict worst case topology and thus be cautious before data transfer is initiated. In our algorithm, at each time-tick, each node decreases the non-zero affinity values of all the links gathered at its information cache by a value equal to the average velocity per time tick. As indicated earlier, affinity is a worst-case prediction about the span of life of a link. So, if the affinity of a link between n and m as perceived by a node a is less than actual affinity between n and m, we will accept the perception of node a about the link-affinity between n and m. We have already defined a metric called Average Link-affinity Convergence in order to evaluate the effectiveness of this scheme.

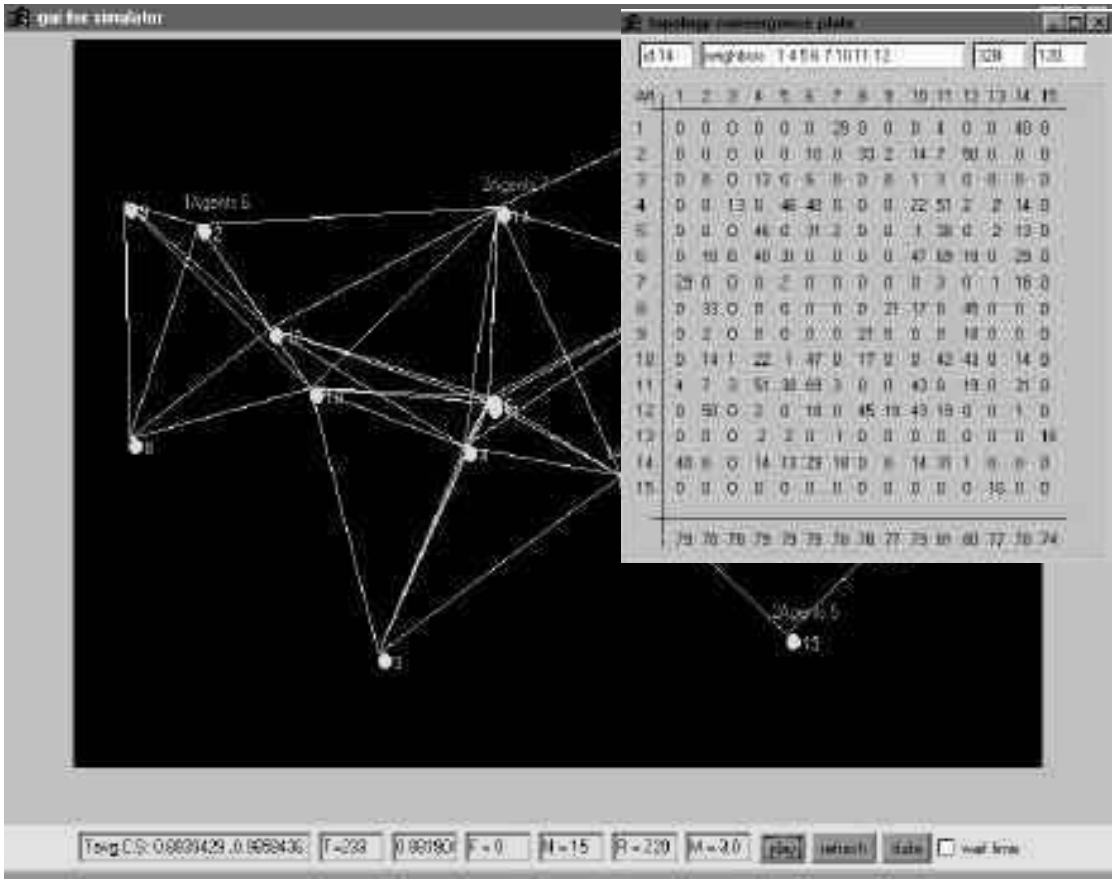


Fig 2. The simulator environment snapshot with N=15 and M=30m/sec.: the nodes move about with agents migrating between nodes. The inset window pops up if a node is clicked. The window shows the information that a node with id=14 possesses about the network in the form of a 15 x 15 matrix, which contains the affinity values of links in 100 msec. unit. The row shown below this affinity matrix displays the **recency token values** of all the nodes in the network. A good navigation strategy ensures that these recency token values have a low standard deviation, implying that the topology is explored homogeneously. The other parameters for the simulation are shown below in the text boxes. The standard deviation of the recency values calculated is = 1.52, mean = 78.26.

7. Performance Evaluation

7.1 Simulation Set-up

The proposed system is evaluated on a simulated environment under a variety of conditions to estimate the connectivity convergence and link-affinity convergence against time. In the simulation, the environment is assumed to be a closed area of 1500 x 1000 square meters in which mobile nodes are distributed randomly. We present simulation results for networks with 30 mobile hosts, operating at a transmission range of 400 m. The bandwidth for transmitting data is assumed to be 1 agents / msec. In order to study the delay, throughput and other time-related parameters, every simulated action is associated with a simulated clock. The clock period (time-tick) is assumed to be one millisecond (simulated). Agent TtM is assumed to be 50 or 100 msec.

The speed of movement of individual node ranges from 5 m/sec to 30 m/sec. Each node starts from a home location, selects a random location as its destination and moves with a uniform, predetermined velocity towards the destination. Once it reaches the destination, it waits there for a pre-specified amount of time, selects randomly another location and moves towards that. However, in the present study, we have assumed zero waiting time to analyze worst-case scenario.

7.2 Evaluating the Basic Mechanism

Fig. 3(a) shows the average connectivity convergence of an ad hoc network for 30 nodes, with 15 agents exploring the network. The agents are allowed to migrate from a node at intervals of 100 milliseconds, i.e. TtM = 100 msec. The transmission range has been assigned as 400 m. This transmission range has been found to give us a more or less connected network. In these cases we see that mobility plays an important role. If the mobility is high, more links information would change per unit time. Thus it becomes more challenging for the agents to carry a link change information to all the nodes in the topology before the next link change takes place. As expected, performance varies directly with mobility. Lower the mobility better the results.

To compensate for increase in mobility, the solution would be to decrease agent TtM. It is evident from the graphs in fig. 3(b) that, for mobility 30 m/s, the performance is significantly different for the high and low TtM. Thus the agents could be adaptive to the average mobility of the nodes and tune its TtM accordingly. Quite obviously, by decreasing agent TtM , we can achieve better convergence. However, a low TtM would imply that nodes get more agents per unit time and thus the network congestion due to agent traffic would also increase. A predictive mechanism on network topology (as discussed in sec. 6) can achieve better result without lowering TtM.

7.3 Predictive Mechanism: Link-Affinity Convergence Analysis

We have analyzed the performance of our prediction mechanism and the results are presented in fig. 4(a) and (b). This mechanism ensures that the node is never misled to believe that a path exists for successful message transfer when the path might actually snap before that actual data is completely transmitted.

The Link-Affinity convergence graphs show satisfactory results as the curve always remains above 98 percent once the topology information has stabilized. This means that a link-affinity information that a node knows about a path has the probability that it is more than 98 % acceptable, even if the agent TtM is 100 msec.

7.4 Discussion

From the results in fig.3(a) it is clear that the average connectivity convergence improves with decrease in mobility. The performance goes below 80 % for a high mobility of 30 m/s. However, the time to migrate (TtM) could be lowered to produce better results even at high mobility (fig.3(b)). But this has the obvious effect of congestion in the system as the system sees more agent traffic in the medium per unit time. Our predictive mechanism (fig. 4) in the context of TtM=100 msec. could yield satisfactory results with the convergence values over 98 %. Thus we see that resorting to the predictive mechanisms even at a low agent migration frequency can uplift performance.

We look here into the congestion introduced in the system due to variation in TtM. Let us assume that agents would take t millisecond to physically migrate from one node to another. Let us assume that our bounded region of ad hoc operation is A sq.mt., our transmission range R, the agent population P and the Time to migrate T msec. In an average case where the topology is evenly distributed over the region A, the number of areas in which agents could migrate between nodes simultaneously, without mutual interference, equals $A / (\pi .R^2)$. Now since the nodes are distributed, the agents would also be found equally distributed in each of these areas in an average case. Thus in any area the number of agents would equal P_a where, $P_a = P.(\pi .R^2) / A$.

As each agent migrates at a time gap of T milliseconds and takes only t millisecond to do so, the medium will be free from agent traffic $[(T - t.P_a) *100 / T]$ % of the time. For example, if the bounded region of operation is 1500 · 1000 sq. m. and R is 400 m, and P and T are 15 and 100 milliseconds respectively for a 30 node network and t = 1msec., then $P_a = 5$ (approx.) and the medium would be free from agent traffic during 95 % of the time. In other words, for only 5 % of the total unit time, the medium gets blocked by agent traffic and the rest is free for data communication processes. This is the real gain in the case of the agent paradigm and serves as the major motivation to replace conventional flooding techniques.

8. Conclusion

In this study, we have designed a mobile, multi-agent based protocol to make the nodes in the network **topology aware.** We have assumed that agents do not get lost

in transit or suffer from any kind of errors in transmission and reception. We have not considered the situations where a set of nodes switch themselves off after creating agents. In this situation, agent population in the network compared to number of active nodes would increase which would degrade the performance. In such a situation, some agents need to be destroyed. We have also not derived analytically the optimal value of TtM under a given condition which would ensure acceptable convergence keeping the agent traffic in the network tolerable. However, our preliminary results indicates that the use of mobile multi-agent framework would be able to make each node in the network topology aware without consuming large portion of network capacity.

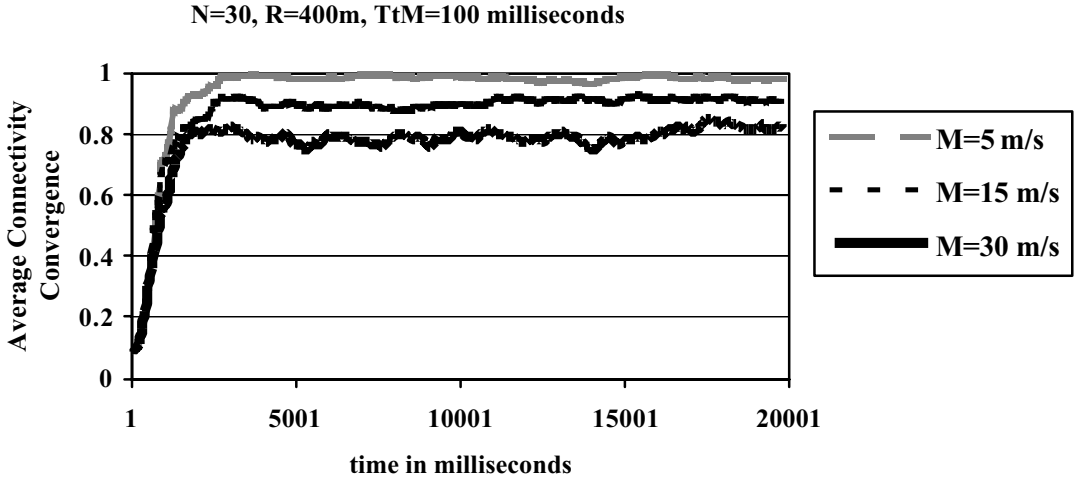


Fig. 3(a). Variation of connectivity convergence with time for different mobility with TtM = 100 msec.

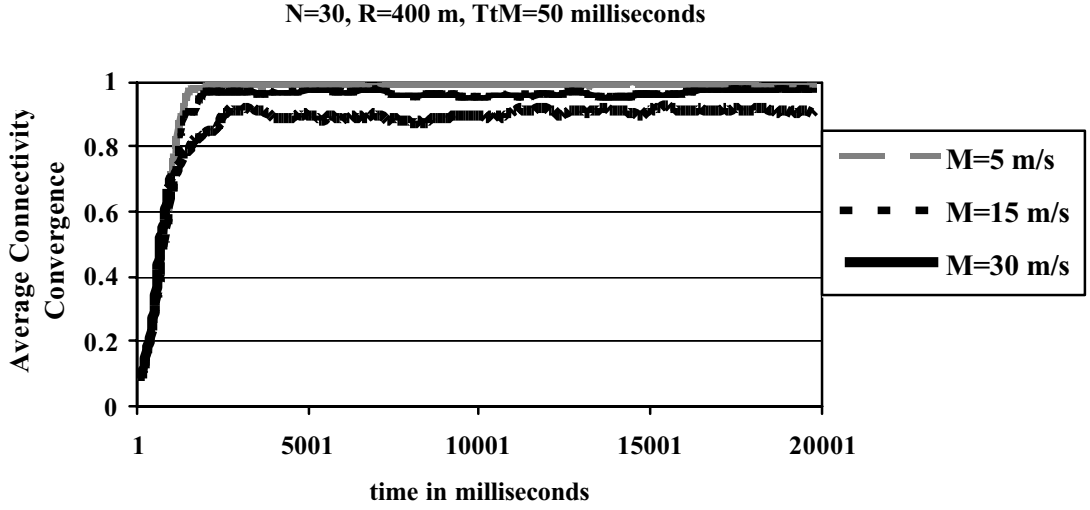


Fig. 3(b). Variation of connectivity convergence with time for different mobility with TtM = 50 msec.

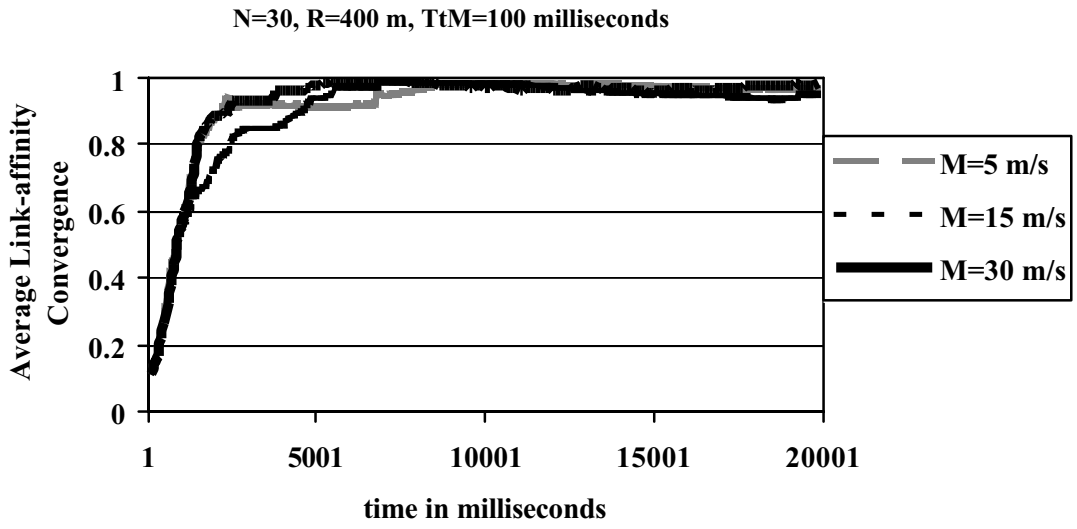


Fig. 4(a). Variation of Average Link-affinity Convergence with time for different mobility with TtM = 100 msec.

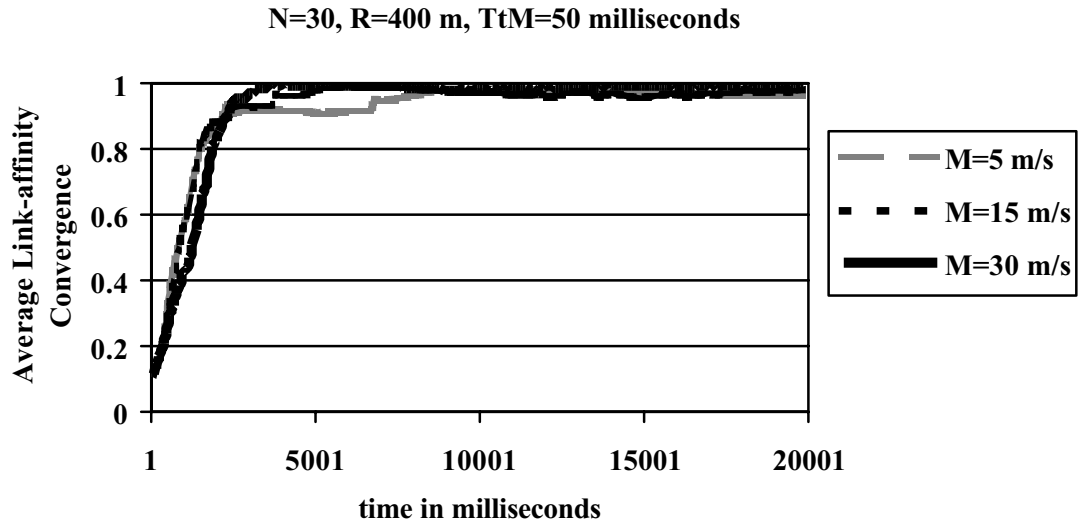


Fig. 4(b). Variation of Average Link-affinity Convergence with time for different mobility with TtM = 50 msec.

References

[1] Vu Anh Pham and Ahmed Karmouch, Mobile Software Agents: An Overview, IEEE Communication Magazine, July, 1998.

[2] R.Gray, D.Kotz, S.Nog and G.Cybenko, Mobile agents for mobile computing, Technical Report PCS-TR96-285, Department of Computer Science, Dartmouth College, Hanover, NH 03755, May, 1996.

[3] D. B. Johnson and D. Maltz, Dynamic source routing in ad hoc wireless networks, T. Imielinski and H. Korth, eds., *Mobile computing,* Kluwer Academic Publ. 1996.
[4] Z. J. Haas, Milcom'97 Panel on Ad-hoc Networks, http://www.ee.cornell.edu/~haas/milcom_panel.html
[5] S. Corson, J. Macker and S. Batsell, Architectural considerations for mobile mesh networking, Internet Draft RFC Version 2, May 1996.
[6] C-K Toh, A novel distributed routing protocol to support ad-hoc mobile computing, IEEE International Phoenix Conference on Computer & Communications (IPCCC'96).
[7] K. Paul, S. Bandyopadhyay, D. Saha and A. Mukherjee, Communication-Aware Mobile Hosts in Ad-hoc Wireless Network, Proc. of the IEEE International Conference on Personal Wireless Communication, Jaipur, India, Feb. 1999.
[8] Steve Appeleby and S.Steward. Mobile software agents for control in Telecommunications networks.*BT Technology Journal*,12(2):104-113,April 1994.
[9] T. Magedanz, K. Rothermel, and S. Krause, "Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?" Proc. INFOCOM'96, San Francisco, CA, 1996.
[10] M. Baldi, S. Gai, and G. P. Picco, "Exploiting Code Mobility in Decentralized and Flexible Network Management", K. Rothermel and R. Popescu-Zeletin, Eds., Mobile Agents, Lecture Notes in Comp. Sci. Series, vol. 1219, pp. 13–26, Springer, 1997.
[11] S. Krause and T. Magedanz, "Mobile Service Agents enabling Intelligence on Demand in Telecommunications", Proc. IEEE GLOBCOM '96, 1996.
[12] Schoonderwoerd, R., et al.(1997) 'Ant-based load balancing in telecommunications networks.' Adaptive Behavior,5(2):169207,1997. http://www.uk.hpl.hp.com/people/ruud/abc.html
[13] G.Di Caro and M.Dorigo.Mobile agents for adaptive routing.In Proceedings of the 31st*Hawaii International Conference on Systems*, January 1988.
[14] Nelson Minar, Kwindle Hultman Kramer and Pattie Maes.Cooperating Mobile Agents for Dynamic Network Routing.In Alex Hayzeldon, editor, *Software Agents for Future Comunications Systems,* chapter 12.Springer-Verlag, 1999. http://www.media.mit.edu/~nelson/research/routes-bookchapter/
[15] S. Bandyopadhyay and Krishna Paul, "Evaluating The Performance Of Mobile Agent Based Message Communication Among Mobile Hosts In Large Ad-Hoc Wireless Networks", The Second ACM International Workshop on Modeling and Simulation of Wireless and Mobile Systems, In conjunction with MOBICOM 99, Washington, USA, August 15-19, 1999
[16] Jonathan Dale. A Mobile Agent Architecture for Distributed Information Management. Thesis submitted for the degree of Doctor of Philosophy. University of Southampton, Department of Electronics and Computer Science.

Mobile Code, Adaptive Mobile Applications, and Network Architectures

Salim Omar, Xinan Zhou, and Thomas Kunz

Systems and Computer Engineering
Carleton University
tkunz@sce.carleton.ca

Abstract. Mobile computing is characterized by many constraints: small, slow, battery-powered portable devices, variable and low-bandwidth communication links. These constraints complicate the design of mobile information systems. In our work, mobile applications, especially ones that do intensive computation and communication (such as next-generation multi-medial PCS and UMTS applications), can be divided dynamically between the wired network and the portable device according to the mobile environment and to the availability of the resources on the device, the wireless link, and the access network. To demonstrate our idea, we developed a code mobility toolkit and experimented with a resource-intense mobile application. With potentially many users executing such applications, the scalability of our approach becomes extremely important. We will briefly discuss performance prediction models based on measurements and LQNs (layered queuing networks). Our results show that it is feasible to support many users with a single dedicated proxy server in the wireless access network.

1. Introduction

Mobile computing is characterized by many constraints: small, slow, battery-powered portable devices, variable and low-bandwidth communication links. Together, they complicate the design of mobile information systems and require rethinking traditional approaches to information access and application design. Finding approaches to reduce power consumption and to improve application performance is a vital and interesting challenge. Many ideas have been developed to address this problem, ranging from hardware to software level approaches.

Designing applications that adapt to the challenges posed by the wireless environment is a hot research area. One group of approaches concentrates on mobile applications that adapt to the scarce and varying wireless link bandwidth by filtering and compressing the data stream between a client application on a portable device and a server executing on a stationary host. Some [3] enhance the server to generate a data stream that is suited for the currently available bandwidth. Others [1,6] extend the client-server structure to a client-proxy-server structure, where a proxy executes in the wireless access network, close to the portable unit. This proxy transparently filters

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 17-28, 2000.

and compresses the data stream originating from the server to suit the current wireless bandwidth.

A second set of approaches provides general solutions that do not change the data stream, focusing on improving TCP throughput [2]. They usually treat IP packets as opaque, i.e., they neither require knowledge of nor do they exploit information about the data stream. While this addresses issues such as high link error rates and spurious disconnections, it does not address the low bandwidth offered by most wireless technologies, nor does it address to problem of limited resources at the portable device.

We propose a third, complementary approach, focusing not on the data stream but on the computational effort required at the portable device. Mobile applications, especially ones that do intensive computation and communication (such as next-generation multi-medial PCS and UMTS applications), can be divided dynamically between the wired network and the portable device according to the mobile environment and to the availability of the resources on the device, the wireless link, and the access network. The access network supports the mobile application by providing proxy servers that can execute parts of the application [16]. This may potentially increase the performance of applications and reduce the power consumption on portable devices since offloading computation to the proxies in the wired network will reduce their CPU cycles and memory requirements [13].

This paper discusses our mobile code toolkit and demonstrates the feasibility of this idea by reporting on our experience with a resource-intensive mobile application, an MP3 player. The results show that both increased application performance and reductions in power consumption are possible under certain conditions by encapsulating the resource-intensive decoding in a mobile agent and migrating it to the less constrained access network.

The paper is organized as follows. Section 2 reviews toolkits to support adaptive mobile applications based on mobile agents/code. Section 3 presents our mobile code toolkit. Some performance improvements and power reductions achievable under certain environment conditions for our MP3 player are the topic of Section 4. Section 5 discusses the scalability of our approach and Section 6 summarizes our main contributions and highlights future work.

2. Related Work

Mobile applications need to be capable of responding to time-varying wireless-QoS and mobile-QoS conditions. Wireless transport and adaptation management systems should therefore be capable of transporting and manipulating content in response to changing mobile network quality of service conditions. Mobile signaling should be capable of establishing suitable network support for adaptive mobile services. In the following sub-sections, we explore some of the major tools and middleware that support adaptive mobile applications.

Comma [8] provides a simple and powerful way for application developers to access the information required to easily incorporate adaptive behavior into their application. It provides easy-to-use methods to access this information, a wide variety

of operators and ranges available to provide the application the information it needs when it needs it, a small library to link with to minimize the overhead placed on the client and to minimize the amount of data that needs to be transferred between the clients and the servers.

The Rover toolkit [7] offers applications a distributed-object system based on the client-server architecture. The Rover toolkit provides mobile communication support based on re-locatable dynamic objects (RDOs). A re-locatable dynamic object is an object with a well-defined interface that can be dynamically loaded into a client computer from a server computer, or vice versa, to reduce client/server communication requirements.

Sumatra [12] is an extension of the Java programming environment. Policy decisions concerning when, where and what to move are left to the application. The high degree of application control allows programmers to explore different policy alternatives for resource monitoring and for adapting to variations in resources. Sumatra provides a resource-monitoring interface, which can be used by applications to register monitoring requests and to determine current values of specific resources. When an application makes a monitoring request, Sumatra forwards the request to the local resource monitor.

Mobiware [1] provides a set of open programmable CORBA interfaces and objects that abstract and represent network devices and resources, providing a toolkit for programmable signaling, adaptation management and wireless transport services. Mobile applications specify *a utility functio*n that maps the range of observed quality to bandwidth. The observed quality index refers to the level of satisfaction perceived by an application at any moment. The *adaptation polic*y captures the adaptive nature of mobile applications in terms of a set of adaptation policies. These policies allow the application to control how it moves along its utility curve as resource availability varies.

In general, it is left to the application to decide how to react to environment changes. This argues for exporting the network state as well as available resources of the portable device to the mobile applications to be designed to be adaptive. On the other hand, the automation of adaptation to the resources was not explored. There are many similarities between our work and the work in Sumatra. Both Sumatra and our work use extended Java Virtual Machines for portability and the ease of use of the language especially for implementing object mobility toolkits. The main difference is that in our work, adaptation to the change in the resources and environment is partially left to the toolkit. We try first to use available remote resources to achieve the same task; otherwise, as last resort, we let the application do the adaptation.

3. Mobile Code Toolkit

The central concept of our framework is the proxy server. An application can use the resources of the proxy server to increase the performance and decrease the power consumption by executing selected objects on the proxy server. To enable this approach, we need to identify computationally intense, closely-coupled groups of

objects and encapsulate them in a mobile agent. This agent may execute locally or be shipped to the access network, depending on the wireless link conditions and the available resources at both the portable device and the proxy server.

The toolkit has a set of APIs, which provide the required functionality for moving objects dynamically. One instance of the toolkit executes on both the portable device and the proxy. It contains the following major modules:

- *Monitor*: monitors and delivers the state of the portable device and the proxy server as events to the Object Server. Changes in the bandwidth or changes in the power status are examples of the events that this unit exports.
- *Code Storage*: storage of the validated classes files (bytecode) at the portable device.
- *Object References and Profiling*: representation of the application's objects along with the profiling information about these objects.
- *Object Server*: core of the toolkit. It runs a thread that listens continuously to all the commands from a remote object server. Commands can be related to moving objects or related to the remote invocation of a method on a remote object.
- *Remote Method Invocation Protocol*: marshal and un-marshal a method's parameters.
- *Dynamic Decision*: analyzes the profiling information of application's objects. It resides only at the proxy server.
- *Communication Control Layer*: simulates wireless links in terms of low bandwidth. We introduce a controllable amount of delay between data packets, which allows us to control the throughput dynamically at run time for testing purposes.

A mobile application needs to be aware of resource availability and changes in the mobile environment. Thus special abstractions are provided to deliver these changes to an application. We model all changes as events, which are delivered to objects. Interested objects in an application must define an event handler through which the events, such as change in the power state and link bandwidth, can be handled.

Our goal is to extend the Java Virtual Machine with a toolkit that facilitates the mobility of objects between portable device and proxy server in a dynamic manner, transparent to the application designers and users. This toolkit is designed to work on PDAs with a limited amount of memory. Other existing ORBs and object mobility toolkits do not support these platforms or they have too big a memory footprint.

To start moving objects of an application between hosts, the notion of a remote reference is required. Java does not support remote references to objects, but it supports the notion of interfaces, which is key to the implementation of a proxy pattern.

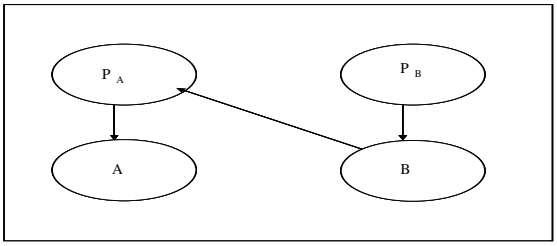


Fig. 1. Proxy Objects with Associated Objects (P_x is Proxy of X)

Every movable application object is associated with a proxy object that has the same interface. Other objects will not reference application objects directly, but they reference them through their proxies, as Figure 1 illustrates. Object *B* references the Proxy of Object *A*, not Object *A* itself.

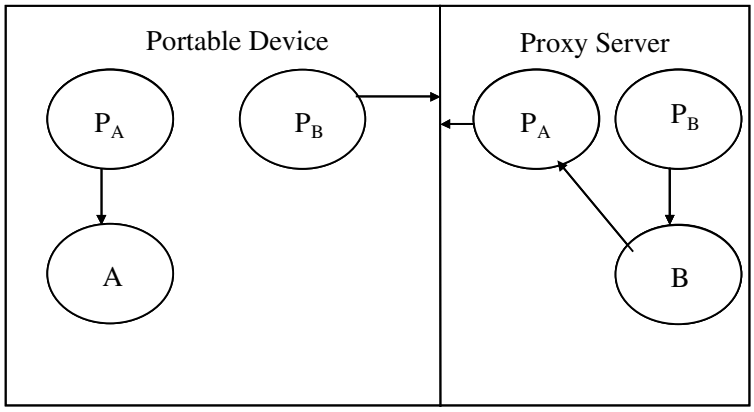


Fig. 2. Migrating Object B to the Proxy Server.

Figure 2 demonstrates moving Object *B* in Figure 1 from the portable device to the proxy server. Moving Object *B* will not require moving Object *A* to the proxy server as well. However, at the proxy server, a proxy of object A must be created to forward the calls to Object *A* at the portable device. Also, a new proxy of *B* will be instantiated at the proxy server to allow local objects there to reference *B*.

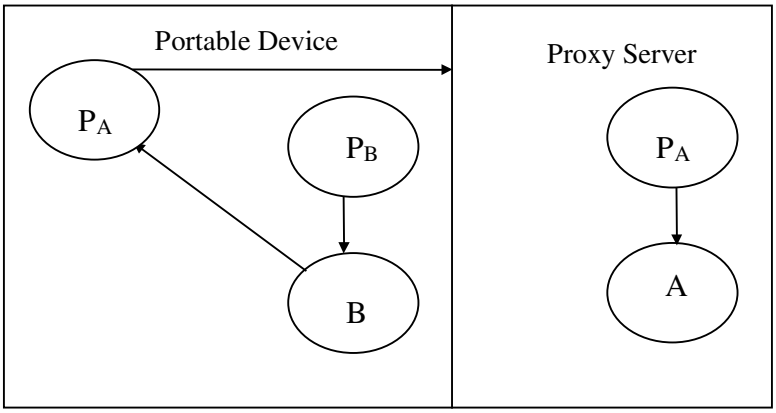


Fig. 3. Migrating Object A to the Proxy Server.

Figure 3 demonstrates migrating Object *A* in Figure 1 to the proxy server. Migrating *A* does not require changing the reference to *A* in *B* since *B* references only the proxy of *A*. Any calls from *B* to *A* will be forwarded remotely through the proxy at the portable device. The proxy of *A* at the proxy server again allows other objects to reference *A* without affecting the flexibility of moving *A* again to the portable device.

Every proxy object created in the toolkit is assigned a local and a remote reference counter. These counters are updated whenever a proxy object is referenced locally or remotely and are used to determine when the garbage collector can claim the proxy and its associated object. Whenever a proxy object is not being referenced remotely

and locally, it will be finalized and garbage collected. If the associated object of this proxy is local, then the associated object will be finalized and claimed again by the garbage collector as well. If the associated object is remote, then the proxy object will inform the remote object server to decrement the remote reference counter for the associated object at the remote site, which in turn garbage collects the object if there are no further references locally or remotely to the associated object.

To obtain a first impression of our toolkit performance, we performed a few simple tests, comparing various aspects with Voyager [11], a popular mobile code toolkit. The following table compares the measured overhead of the toolkit against Voyager. The measurements were taken under Windows NT on a Pentium II 350 MHz PC.

Table 1. Overhead Comparison

	Voyager	Our toolkit
Footprint	2620 KB	204 KB
Moving object	142 ms	80 ms
Calling a method	23 ms	110 ms

Based on these results, we are confident that our toolkit can be used for small portable devices such as Palms and PDAs. The memory requirement of our toolkit is small compared to Voyager (which supports many additional features such as CORBA compliance), allowing it to be embedded in these small devices. The cost of migrating the objects compared to Voyager is lower as well; however, we still need to improve the remote method invocation protocol.

4. Case Study

We implemented an MP3 player in Java to demonstrate the feasibility of our general approach. This application requires a powerful CPU to decode the sound file due to the complexity of its encoder/decoder algorithm, which makes it an ideal candidate to demonstrate the need for fast static hosts, i.e. proxy servers, to support the relative resource-constrained portable devices.

We executed the MP3 player under various emulated environment conditions and observed application performance and power consumption on the portable device. Based on the observed environment, our runtime system instantiates some objects on the proxy server, others are created on the portable device. We studied in particular the following parameters:

1. Available Bandwidth.
2. Relative CPU speeds (Portable Device: Proxy Server).

To observer the importance of the first parameter, the bandwidth available, we choose 19.2 Kb/sec to represent CDPD [4], a typical wide-area cellular data service. For high bandwidths we choose 1000 Kb/sec to represent the bandwidth that can be obtained from Wireless Ethernet cards such as WaveLan [10].

To observer the importance of the second parameter, the relative CPU speed, we fixed the bandwidth to 1000 Kb/sec, so it does not represent a scarce resource. A Windows CE PDA and a laptop were used as portable devices. The PDA runs a RISC

processor of 75 MHz and the laptop runs a Pentium processor of 133 MHz. The proxy server runs on a 350 MHz Pentium II PC. The performance of Java applications depends primarily on the performance of the JVM. Both laptop and the proxy server run high performance JVMs. The JVM on the PDA, on the other hand, is very slow, so the relative CPU speed degrades considerably. We measured the relative CPU speed between the PDA, laptop and the proxy and found it to be 1:116 and 1:4, respectively.

The experiments are based on decoding MP3 coded audio frames, with the assumption that output is mono, with a sampling rate of 11025 Hz, and 16 bits per sample (which will impact the network traffic when decoding is done remotely and is based on the achievable quality of sound-cards in low-cost handheld devices).

The detailed experiments and results are discussed in depth in [14]. These results demonstrate that available bandwidth is an important factor. If bandwidth is the bottleneck in the system, neither reduction in power consumption nor increases in MP3 player performance can be obtained, no matter what the relative CPU speed is. However, if bandwidth is not the bottleneck, then the relative CPU speed becomes a decisive factor in increasing the performance and decreasing the power consumption at the portable device. It is possible to save power and increase performance of the MP3 player if the entire decoder will be executed remotely and the PDA only works as sound player. These improvements can be very dramatic: the application will execute up to 20 times faster if decoding is done at the proxy, consuming only 5% of the power it would take to decode the MP3 file on the PDA. On the other hand, using the laptop as portable device, an MP3 player that decodes locally always performs better than decoding at the proxy, even though the available bandwidth is sufficient to handle the decoded sound and the computational power is quit high at the proxy server.

Overall, the results show that it is not always beneficial to ship agents to more powerful proxies to gain performance and/or decrease power consumption. The benefits depend on available bandwidth and relative CPU speed. We also expect them to depend on the graph topology and the data traffic volume between application objects. For our sample application, an MP3 player, migrating the decoding component to a more powerful proxy leads to a considerable decrease in power consumption as well as an increase in the performance when executing on a Windows CE PDA; however, it is not worth migrating the MP3 decoder to a proxy server when using a laptop as portable device.

5. Scalability

To support our approach, proxy servers need to be deployed throughout the access network, which could be a large provincial or national cellular network. At the one hand, one could envision an architecture where each wireless cell provides dedicated proxy servers, resulting in relatively little concurrent use of an individual server but inducing a high handover overhead and costs. On the other extreme, we could provide only one or very few proxy servers that support applications in many different wireless cells, reducing the handover overhead but requiring more powerful servers.

With potentially multiple thousands of users executing resource-intensive next-generation mobile applications, the scalability of our approach becomes extremely important. To explore this issue, we started to develop performance prediction models based on LQNs (layered queuing networks).

Layered Queuing Networks study the performance of distributed systems that have hardware and software [5,15]. A task is a basic unit in LQN. A task represents the software in execution. An entry represents a service provided by a task. If a task can provide multiple services, then the task has multiple entries. Entries in different tasks communicate with each other through requesting and providing services. Client tasks make requests to proxies, these in turn invoke services provided by the application server task. Requests are either synchronous or asynchronous.

Each task has at least one entry. The entries in a task have execution demands respectively, and may also interact with entries in other tasks by calling the entries in those tasks. The client tasks will not receive requests from other tasks. They are called reference tasks. For reference tasks, usually there is a think time that is denoted by Z, which implies the pause time between two consecutive operations. Execution demands of entries are characterized in two phases. Phase 1 is the service demand between reception and response (for synchronous requests), phase 2 describes the service demands after the response (for synchronous requests) or the total service demand for asynchronous requests. The LQN analytical tool describes the system by the average behavior of the entries and solves the performance model by approximate MVA calculations.

To study the scalability of our system, we developed a four layer LQN, extracted data from traces collected from an operational WAP-based system [9], and studied the impact of introducing proxy servers. In modeling terms, the introduction of a proxy results in less execution demand on the portable devices and more execution demand on the proxy servers. Since we assume the proxy servers to be more powerful, the increase in load is only a fraction of the load decrease on the portable device.

The complete model is shown in Figure 4. A parallelogram represents a task entry. Cascaded parallelograms indicate an entry of multiple tasks. The task name is written near the parallelogram. [Z] in the client task entry models the client think time. [0, tc] in the client task entry represents the execution demands of the client task entry between requests. The pair of brackets inside the non-referential task entries has the same meaning as the one in the client task entry. The notation under the pair of brackets is the entry name. The ellipse represents CPU processors. The arrow segment connects the calling entry and the called entry. The straight segment connects the task and the CPU on which the task runs. The pair of circular brackets beside the arrow line contains the number of calls from the calling entry to the called entry. ‘sh’ denotes synchronous calls and ‘ay’ denotes asynchronous calls. Client tasks make (indirectly) requests to an application server and wait for the responses. This server answers some of the requests directly and passes some to other servers on the Internet. Generally, passing a request to another server takes less time than answering one directly. The application server task has four entries. The first entry *se1* processes the synchronous requests from client tasks and responds to the clients directly. The second entry *se2* is responsible for asynchronous requests from client tasks. The third entry *se3* passes synchronous requests from the clients to other servers. The *General Server* task is used to represent additional servers on the Internet since it is impossible

to get information for all the Internet servers and model them individually. The fourth entry *se4* is used to represent the idle time between consecutive sessions with the help of an imaginary *Idle Server* task and CPU4.

We studied the capacity of the system under various conditions and the effect of transferring execution load from handsets to proxies. The capacity of the system indicates the maximum number of users that the system can serve at the same time without being saturated. Proxies, Application Server, General Server and Idle Server are multithreaded, and can provide as many threads as needed. We define 0.7 as the threshold utilization of CPU2, beyond which we consider that the system is saturated. 'MC' is the maximum number of clients that the system can sustain in this case. We studied the effect of increasing (the percentage of requests serviced directly by the application server and executing a higher portion of the client tasks at the proxy servers.

The capacity of the system decreases with the increase of *sh1*. That is, the more requests the application server processes directly, the smaller the system capacity becomes, as expected. If the application server processes all requests, the system can only support 8 concurrent clients. If all requests are forwarded to other servers, the system can support up to 50 concurrent clients. In our trace files [9], we found that the average maximum numbers of concurrent sessions are indeed typically below 8. Once the average number of concurrent sessions exceeded these numbers, and we received verbal confirmation the traced system experienced performance problems. This indicates that such performance prediction models can indeed be useful in anticipating capacity problems.

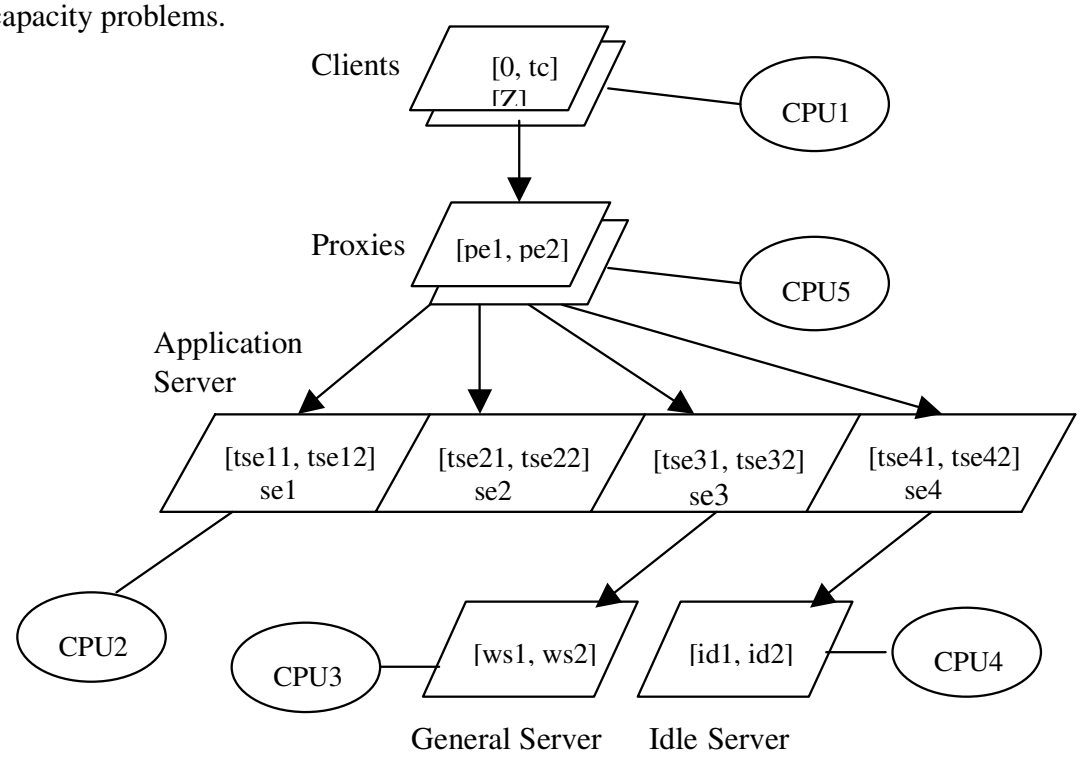


Fig. 4. Layered Queuing Model

We also investigated the effect of load migration from handsets to the proxies. We assume that the proxy CPU is 25 times faster than the handset CPU. The load

migration from handsets to the proxies reduces the service demand at the clients' side (which is equivalent to making a slow user fast). This will reduce the capacity of the system. We varied the service demand on the portable device (*tc*) from 6 to 0, in steps of 1, with a corresponding (smaller) increase in service demand (*pe1*) at the proxy. The performance prediction results are shown in Tables 2 and 3.

Table 2. Capacity vs. Load Migration, all client requests processed by application server.

tc	6	5	4	3	2	1	0
pe1	0	0.04	0.08	0.12	0.16	0.2	0.24
MC	8	7	7	7	7	6	6

Table 3. Capacity vs. Load Migration, all client requests forwarded to external servers.

tc	6	5	4	3	2	1	0
pe1	0	0.04	0.08	0.12	0.16	0.2	0.24
MC	50	46	43	42	41	40	39

We can see that, all else being equal, the capacity decreases with increasing migration of computational load from portable devices to the proxies. This is consistent with other result reported in [17] that show that the system can serve more slow users than fast ones.

6. Conclusions and Future Work

Finding approaches to reduce power consumption and to improve application performance is a vital and interesting problem for mobile applications executing on resource-constrained portable devices. We suggested a new approach in which part of an application will be encapsulated in a mobile agent and potentially shipped for execution to proxy servers, according to the portable device and fixed host's available resources and wireless network state. To support our approach, we designed and developed a mobile object toolkit, based on Java. With this toolkit we combine JVMs on both the proxy server and the portable device into one virtual machine from the application point of view. The results showed that it is possible to simultaneously improve application performance and reduce power consumption by migrating the entire MP3 decoder to the proxy server in the case of a slow portable device and sufficient wireless bandwidth.

To study the scalability of our approach, we developed a Layered Queuing Model, derived trace data from a live system, and studied the maximum number of concurrent users that can be supported. Even in a system with many potential users, our traces reveal that only a relatively small number of users are concurrently accessing the application server. In these cases, the introduction of proxy servers does not overly reduce system capacity. Other studies have shown that for the system studied, the application server is more likely to become a bottleneck, rather than the proxy server. While these findings are application-specific, they are encouraging. Contrary to our initial suspicions, we will not need proxy servers in each cell to support the user

population. In all likelihood, a few, centrally placed, proxy servers can support potentially many users.

A number of issues need to be addressed in future work, some of which is currently under way. We are working on improving the mobile object toolkit. The main improvement to our toolkit optimizes the RMI protocol. Another improvement deals with proxy objects. To support location-transparent invocation of methods, each object is associated with one or more proxy objects. Currently, we manually write these proxy objects; however, we plan to develop tools to automate this process (similar to Voyager) and integrate it with the toolkit. Finally, we will plan to port the toolkit to Palm OS.

A second issue is to explore scenarios where either the application behavior or the execution environment changes drastically while the application executes. Intuitively, we would expect the runtime system to rebalance the application accordingly. However, migrating agents/objects at runtime is not cheap. So we need to explore how to balance the resulting overhead with the anticipated performance gains and power reductions, in particular in execution environments that change rapidly. Finally, while our results reported here show that there is no trade-off between power reduction and performance improvement, previous work [13] indicates that there may well be such trade-offs for other applications. In these cases, we need to identify how to balance conflicting objectives. One possible solution could be to allow the mobile user to select preferences that prioritize objectives.

A third area is the refinement of the performance prediction model. The current model is essentially based on a multi-tier client-server architecture. In cases where the mobile agent acts like a server (decode the next MP3 frame and return the sampled sound), this accurately reflects the application structure. In more general cases, though, objects at the proxy server side will request services from objects that remain at the portable device. Another refinement of the model would include the wireless link as additional "service" layer between client tasks and proxies, to capture contention for that shared and scarce resource.

A final area of possible future work is the interaction between application-aware and application-transparent adaptation. Our MP3 player does not react to changes in bandwidth, for example by reducing sampling size or audio quality. In our experiments, we fixed the output playing rate and the sampling size. Further study is required to show how application adaptation policies affect and interact with the automated adaptation by our toolkit.

Acknowledgements

This work was support by the National Research and Engineering Council, Canada (NSERC) and a research grant by Bell Mobility Cellular, a Canadian wireless service provider.

References

1. O. Angin et al. *The Mobiware toolkit: Programmable support for adaptive mobile networking*. IEEE Personal Comm., 5(4), Aug. 1998, pages 32-43.
2. H. Balakrishnan et al. *Improving TCP/IP performance over wireless networks*, Proc. of the 1st Int. Conf. on Mobile Computing and Communications, Berkeley, USA, November 1995, pages 2-11.
3. J. Bolliger and T. Gross. *A framework-based approach to the development of network-aware applications*. IEEE Trans. on Software Eng., 24(5), May 1998, pages 376-390.
4. CDPD Consortium, *Cellular Digital Packet Data System Specification,* Release 1.1, January 19, 1995 (CD-ROM).
5. G. Franks and M. Woodside, *Performance of multi-level client-server systems with parallel service operations*, Proc. of the 1st Int. Workshop on Software and Performance (WOSP98), Santa Fe, October 1998, pages 120-130.
6. A. Fox et al. *Adapting to network and client variation using infrastructure proxies: Lessons and perspectives*. IEEE Personal Comm., 5(4), Aug. 1998, pages 10-19.
7. A.D. Joseph et al. *Rover: a toolkit for mobile information access*. ACM Operating Systems Review, 29(5), Dec. 1995, pages 156-171.
8. D. Kidston et al. *Comma, a communication manager for mobile applications*. Proc. of the 10th Int. Conf. on Wireless Communications, Calgary, Alberta, Canada, July 1998, pages 103-116
9. T. Kunz et al. *WAP traffic: Description and comparison to WWW traffic*. Proc. of the 3rd ACM Int. Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, (MSWiM 2000), Boston, USA, Aug. 2000.
10. Lucent Technologies. WaveLAN Wireless Computing, http://www.wavelan.com/.
11. ObjectSpace. Voyager 2.0.0 User Guide, http://www.objectspace.com/Voyager/.
12. L. Ranganathan et al. *Network-aware mobile programs*, Dept. of Computer Science, University of Maryland, College Park, MD 20740.
13. S. Omar and T. Kunz. *Reducing power consumption and increasing application performance for PDAs through mobile code*. Proc. 1999 Int. Conf. on Parallel and Distributed Processing Techniques and Applications, Vol. II, Las Vegas, Nevada, USA, June 1999, pages 1005-1011.
14. S. Omar. *A mobile code toolkit for adaptive mobile applications*, April 2000, Thesis (M.C.S.), Carleton University, School of Computer Science.
15. J.A. Rolia and K.C. Sevcik. *The method of layers*. IEEE Trans. on Software Engineering, 21(8), August 1995, pages 689-700.
16. J. Wang and T. Kunz. *A proxy server infrastructure for adaptive mobile applications*. Proc. of the 18th IASTED Int. Conf. on Applied Informatics, Innsbruck, Austria, February 2000, pages 561-567.
17. X. Zhou, *Cellular data traffic: analysis, modeling, and prediction*, Master's Thesis, School of Computer Science, Carleton University, July 2000 (expected).

Toward a Mobile Work Environment

Larbi Esmahi[1], Roger Impey[1],
Ramiro Liscano[2]

[1] Institute for Information Technology,
National Research Council of Canada,
Ottawa, Ontario, K1A 0R6, Canada
{Larbi.Esmahi, Roger.Impey}@nrc.ca
[2] Mitel Corporation, Strategic Technology
350 Legget Drive, P.O. Box 13089
Kanata, Ontario, K2K 2W7, Canada
Ramiro_Liscano@Mitel.com

Abstract. The growth of Internet and the deregulation of the telecommunication industry have exploded the number of tasks and services that we can do or obtain from our terminals (PCs, Laptops, PDAs and Phones). Hence the desktop of our terminals has became the control board for almost all of our works. This paper addresses the problem of how to make this work environment available for roaming users with the same "look and feel" across different network, and from different terminals. In this paper, we propose a solution based on the concept of virtual home environment and agent's technology. We describe the architecture for implementing this mobile work environment and we give some arguments that justify the use of mobile intelligent agents for implementing this framework.

1 Introduction

During the last century the office environment (i.e. working environment for leaders, managers, or officers) has been continuously evolved, from simple calendar on paper, passing by the integration of typewriter and telephone, for finally arriving to a complex environment that integrate an arsenal of services and electronic gadgets (e.g. electronic agenda, calendar, mobile telephony, PDA, Laptop, PCs). Until now a little part of this environment is mobile (completely mobile like universal mobile phone or partly mobile like some Internet services and E-mail).
The need for providing an ***Integrated Mobile*** home environment arises because of two major developments: one sociological and the other technological.

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 29-48, 2000.

- *In the sociological aspect*, the development of Tele-work, Tele-teaching and telecommunication facilities imply that the most of the home environment users (e.g. managers, officers, students and researchers) becomes increasingly nomad.
- *In the technological aspect*, one of the main goals in the standardisation of 3rd generation mobile networks [1] is to provide a framework for service creation and provision, rather than a complete set of services as done in the traditional approach[1]. A consequence of this approach is that the same services may not be available to a user when they have roamed into another network or at least it may not be available in a flexible "look and feel" manner.

To overcome these problems, the concept of *Virtual Home Environment (VHE)* [2] was introduced into the standardisation process for the deployment of personalised services across network and terminal boundaries with the same look and feel.
The scope of the VHE covers also the provision of users with consistent access to services from different terminals, and with the ability to personalise services so that as far as is practicable the individual needs of the user are met.
The ability to provide such flexibility service deployment has many implications for the management of services, the control of services, the signalling process and the rooting process. For example, a user's request for a non-locally available service will require some means to locate where that service request can be handled or where information can be found on how to handle the request itself. Hence, such flexibility implies that service management and control is distributed in the system, giving rise to complicated network management protocols and stringent security requirements.
In the last few years, agent technology is continuously evolving in many areas [3], including user interface, personal assistance, information retrieval, E-commerce and telecommunication services management. This new technology, more especially mobile agents offer a promising solution to cope with the complexity of open environments.

Definition:
*The **Virtual Home Environment** is defined as a concept for personalised service portability across network boundaries and between terminals. The concept of the VHE is such that users are consistently presented with the same personalised features, user interface customisation and services in whatever network, whatever terminal (within the capabilities of the terminal), wherever the user may be located [4].*

Hence, the main goals of the home environment system are:

- *Personalised services*: providing the user with the facility to access, subscribe and personalise the available services from any network or terminal according to the provider's limitations and subject to network and terminal capabilities.

[1] Everywhere you roam in the world you find the same phone terminal with approximately the same functionalities.

- *Personalised interface*: providing the services to users with the same "look and feel" irrespective of the network type and within the limitation of the network and terminal.
- *Global service availability*: providing the user with his subscribed services across network boundaries when roaming.
- *Enabling the creation of new services*: the creation of new services is enabled by providing access to service features and service capabilities features by means of generic interfaces. Those generic interfaces should be: (1) independent of vendor specific solution, (2) independent of the location where service capabilities are implemented, (3) independent of supported server capabilities in the network and (4) independent of programming languages of service capabilities.

In this paper we propose an architecture for implementing a mobile work environment using the concept of virtual home environment and mobile intelligent agent (MIA) technologies [5]. In section 2, we define the VHE functionalities. In section 3, we explain the parity between VHE and MIA by giving different arguments that made the MIA technology more suitable for implementing the mobile work environment. In section 4, we describe the architecture for implementing the mobile work environment. Finally, in section 5 we conclude with some of our future works related to this project.

2 VHE Functionalities

In order to follow the goals sited on top, a VHE system has to perform the following tasks:

- User authentication: a universal user identity must be maintained. This will allow the user to be authenticated from everywhere and using different types of terminals.
- User profiles management: allowing interfaces and services personalisation, and synchronising different user profiles with the central register.
- Service management: managing the service subscription/unsubscription, access, QoS negotiation and security.
- Service interaction: handling interaction events and communication between subscribed services. Other tasks that imply co-ordination of different services process such as unified billing should also be handled.
- Business management: contract negotiation between subscriber and provider. Co-operation contract between different providers for multi-parties services.

Two main components interact in the VHE environment in order to fulfil those tasks. The basic services (the universal profile and the universal user identity) that constitute the core of the VHE system and the set of other available services announced by different providers.

2.1 Basic Services

2.1.1 Universal Profile

The *Universal Profile (UP)* service enables the user to access the available services with specific features according to his interests and available user's interfaces, (e.g., the user should be able to choose unified messaging features and real-time stock quote information for Internet email and GSM user's interface). The UP service should support the following functionalities:

- Provide the user with a single unified interface for all services (e.g. unified messaging, agenda, VPN and personal information) and operations (e.g. activate, deactivate, subscribe and unsubscribe).
- Enable the user to define and manage services features, interfaces features and terminals features. This management should be done from different interfaces and terminals.
- Detect features conflict on profile specifications and report them to the user.
- Enable the user to establish rules for use and availability for each feature.
- Provide multilevel profile management: in a first level, corporate customers may require to manage their users (add/remove users, modify access constraints) and to have functionalities like statistics. In a second level, each user should have his individual user profile containing personal information and service specific information in form of a service profiles and terminal profiles.
- Wrap the chosen features in a package and provide them as one single service that the user can access in a location and network independent manner.

2.1.2 Universal User Identity

The *Universal User Identity (UUI)* service provides a unique user's identity that is valid for any subscribed service. The UUI service should support all user's identifications (e.g. phone number, e-mail address, fax number, UUI) to reach the user from any supported user's interface and subscribed service, independently of the available user's interface destination and service. The UUI service should support the following functionalities:

- Allows the user to manage his identity independently of the used interface, terminal or connection.
- Select the interface and service to use for reaching the user according to the UP rules.
- Provide a UUI directory through which it is possible to get UUI addresses.

2.2 Other Services

The VHE environment is increasingly including a set of communication and information services from which we can enumerate the more used ones:

Unified messaging: a service that provides the dispatch, notification, retrieval and management of any valid type of message (e.g. email, fax, voice-mail, GSM/SMS) on any supported users' interface (e.g., Internet HTTP/VoIP, PSTN/GSM Telephony).

- *Unified notification*: the unified notification feature supports the delivery of short messages coming from any subscribed service, at any valid time and at any users' interface.
- *Personal information service*: a service that provides information according to subscriber's profile, at the most appropriate time and user's interface. There should be a *Content Model* and a *User Model* to relate information and subscribers' interests, automatically.
- *Advanced agenda service*: a service that provides the subscriber with real-time notifications about registered appointments through subscribers' most convenient way, (i.e., by Paging, SMS, IP or PSTN/GSM Telephony, email or Desktop Tickers/Screensavers).
- *Virtual Address Book*: a service that allows the user to manage his personal address books through a variety of terminals, and across different networking environments. It also allows keeping the main address books up to date using an automatic transfer of data from other copies stored in different terminals (e.g. Laptop or home PC).
- *Enhanced virtual private network service*: the VPN service can be described as the ability to "tunnel" the internet or any other public network in a manner that provides the same security and other features formerly only available on private networks [6]. The enhanced VPN service should support in addition of traditional features (call restriction, secure access, abbreviated dialling and unified billing) other features such as integration of several terminals (e.g. fixed and mobile phones, PCs, PDA) and several communication modes (e.g. phone-to-phone, phone-to-PC, PC-to-phone, PC-to-PC)
- *Click-to-Dial*: a user is able to initiate a telephone call by clicking a button during a web session. The called address may be either an IP address or a phone line number.
- *E-commerce service*: a service that provides a virtual market place with a well-defined structure for E-commerce transaction. The service should provide a communication mechanism that support contract negotiation and delivery of products, and implement a mediator and broker services. The mediator service assists the consumer and the provider in getting a contract and the broker service play a matchmaker role between them.
- *Flexible financial service*: a service that allows the user to access his banking services and conducts secure transactions anytime and anywhere.

3 Why Mobile Intelligent Agents for VHE?

The main objective of the VHE is to enables individually subscribed and customised services to follow theirs associated users wherever they roam. So, it is interesting to

consider the introduction of mobile agents to enable this dynamic service provisioning and management. The agents' technology brings several advantages for implementing new services on distributed systems. In fact, they allow distributing the functionalities in small, reproducible and distributed software entities. They also allow for a clear and easy separation between their internal, private knowledge, and their interface towards the external world and other agents.

The VHE service provisioning and management fits better for exploiting the agents' properties.

Agents' autonomy:

- Allows making decision on service access, interfaces' configuration, and service provisioning without human assistance.
- Allows automating control and management tasks, hence relaxing the operators' workload.
- Allows automating service deployment and provision and thus reducing the required effort and time for the installation operation and maintenance of services.

Agents' intelligence:

- Allows the dynamic customisation and configuration of services. The agents can learn and adapt to the preferences of their users and detect and updates old versions.
- Allows supporting the principle ideas behind the VHE services. That is, to allow service intelligence to be downloaded dynamically from providers and to allow service intelligence to be distributed between different providers.

Agents' mobility:

- Supports the dynamic topological of service provisioning.
- Enables telecommunication services to be provided instantly and customised directly at the locations where the service is needed.
- Enables dynamic provision of customised services by downloading service agents from the service provider system to the network nodes or user terminals.

Agents' sociability:

- Offers the potential to distribute service related processing, and a mechanism for nodes in different networks to co-operate in order to provide a service to the user.
- Allows negotiation of service features.
- Provides multi-services interaction and co-ordination.
- Allows asynchronous and co-operative processing of tasks. In fact, the possibility for delegating tasks to nodes using MIA allows for highly dynamic and parallel computations.

The agents' technology fits better for VHE implementation, since it supports the following requirements:

Dynamic scalability:

- MIA supports huge distributed systems such as the Internet. In fact, each service is modelled as a collection of agents, each agent occupying different places at different times since it can move from one place to another.
- MIA supports on-demand provisioning of services. In fact, when servers are implemented using MIA, the agents' mobility allows to deploy new replicas when the demand arises or to migrate to the location where the demand is high.

- MIA enables the provision of flexible solutions, where services are partitioned into mobile service agents realising dedicated functionalities that can be spread across the network.

Distribution of services:

- MIA fit better for modelling the ideal situation for a mobile user. Since, mobile agents can implement the ubiquitous availability of applications, data files and user profiles by using the concept of mobility and cloning.
- MIA enable control tasks to be performed in a real distributed manner by using the concept of remote programming [7] in stead of the client/server programming concept used currently in intelligent network [8].
- The possibility to bring control or management agents as close as possible to the resources, allows for a more decentralised realisation of service control and management software.

Reduction of traffic:

- MIA allows decreasing pressure on centralised network management systems and network bandwidth by using both spatial distribution and temporal distribution.
- MIA autonomy and asynchronous operation allow reducing the requirements regarding traffic load and the availability of the underlying networks

Independence regarding failures:

- MIA allows reducing the influence of signalling network fault during service processing, since once a service agent has migrated, the processing will be performed locally.
- The agents' migration to the required data allows reducing dependence regarding network availability. So, more robustness is achieved for distributed system.

This new technology offers a promising solution to cope with the complexity of open environments, since agent-based solutions can be used in the following areas of the VHE system.

- Usage of agents for value added service subscription and provision to the user, such as dynamic migration of applications between the user's mobile terminal and the value added service-provisioning operator. A mobile software agent transports the software to the current terminal and executes it there.
- Usage of agents for providing the user with the same "look and feel" across different networks and using different terminals. The idea is to provide customised communication capabilities dynamically to the user and increase the intelligence of the terminal.
- Usage of agents for user and service roaming. A mobile software agent can follow the roaming user, even between different mobile communication systems, to represent the user in the foreign network and provide the user's subscribed services.

Finally, it has to be stressed that agent technology offers a number of very interesting advantages, but it should not be seen as the only solution of all communication environments. It should rather be seen as a technology that comes to resolve some situations. Furthermore, we have to consider some of MIA disadvantages that are:

- Requirement of a specific run-time environment (agent execution environment) to be present in all nodes to be visited
- The security problem aspect. The platforms have to be protected from malicious agents and vice versa.
- The network load that may increase in some situations. One of the mobile agents' goal is to reduce the network traffic, but it does not seem useful for every agent to migrate in every situation; this would probably increase the network traffic. Therefore, new strategies have to be developed to establish under which circumstances an agent will migrate.
- The MIA technology doesn't provide location transparency. Each agent must be aware of the location to be visited.

The MIA technology is relatively new and its suitability for solving telecommunications problems needs to be proven in implementations.

4 VHE Implementation Scenarios

Our implementation will use a modular architecture based on mobile and fixed intelligent agents. For services that we decide to integrate to our system and are available on the market we will develop intelligent interfaces that allow us to integrate them according to the VHE concept.

As we have stated in section cf. 3, the VHE system is composed of two types of services, basic services and other services. The core of the VHE is defined by the basic services that are common services used by all other services.
In the multi-agent system used to implement the VHE services, we distinguish two levels: (1) the architecture of the whole system that defines how the agents communicate between themselves, co-ordinate their activities, negotiate and provide services, and (2) the internal architecture for each agent. Both these two levels need to be evaluated for the most suitable implementation architecture for the VHE system.

4.1 The Platform Architecture

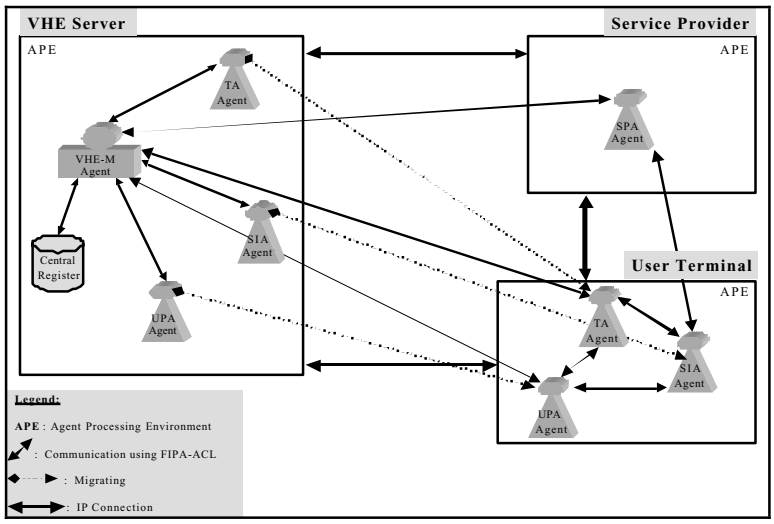


Fig. 1. Architecture for the proposed VHE

4.1.1 VHE Manager

The VHE Manager (VHE-M) is the most essential agent of the VHE system. It stores and manages the knowledge related to the user and all available services. The main tasks of the VHE-M are:

- Performing user authentication
- Acting as a central register, where each new specific service must be registered. This is performed by the communication between the VHE-M and the service provider agent (SPA) representing the service.
- Managing and assuring the consistence of the database containing user and service profiles.
- Receiving service requests from terminals and giving access to service capabilities and user profiles data.
- Instructing the service interface agent (SIA) and the user profile agent (UPA) on how to display the service on the terminal according to all relevant information in the profiles.
- Initiating and sending the SIA to display the required service on the corresponding terminal. This is done in co-operation with the SPA. In fact, the services could be implemented on different platforms running on different operating systems.
- Checking the version of the terminal agent (TA) that resides on remote terminals and automatically downloading the newest one.

The VHE manager is implemented as a stationary agent residing in specific place (fixed address) known by all other agents. Three type of connection can be used to contact the VHE-M: using a web page, using a telnet connection or using a dial-up connection.

4.1.2 Terminal Agent

The terminal agent (TA) moves to the terminal for providing an interface to the user for authentication and service request. However, for terminals that haven't the ability to accommodate the TA, this later resides on the VHE server and communicates with the terminal using a gateway. The main tasks performed by the TA are:

- Providing an interface to the user for authenticating himself and requesting services according to the terminal specifications.
- Maintaining the user terminal profile of the corresponding end-device considering its capacity and system features.
- Communicating with the VHE-M to get information about available services, and for addressing the user requests.
- Receiving the service interface agent and starting it.
- Providing the user with all functionalities available on the VHE system according to the specific terminal features.

The TA can be viewed by the user as a desktop application, which sets up, updates and presents the VHE service links. The TA is also used to browse available services, subscribe/unsubscribe and start services. The TA is linked to profile information by communicating with the UPA and SIA.

4.1.3 Service Interface Agent

When a user request a service and is authenticated having access to this service, the VHE-M in co-operation with the service provider agent instructs the activation of a service interface agent (SIA) for presenting the service. Depending on the type of the terminal used, the SIA migrate to the terminal or only start a communication with the terminal using a gateway. The main tasks of the SIA agent are:

- Managing the communication between the terminal and the SPA during the service session.
- Displaying the service according to user preferences and terminal capabilities.
- Reporting modifications done by the user on his preferences to the agent profile.
- Reporting specific data such as time and statistics of use to the TA, which communicate them to the VHE-M agent.
- Offers the possibility to download a demo version of the service when it is available.
- Provide on-line assistance on the service.

4.1.4 Service Provider Agent

The service provider agents (SPA) are agents that represent a specific service to be provided. A service provisioning usually consists of a service provider agent and a service interface agent. The SAP resides either on the provider server or on the VHE server and must register with the VHE-M. The main tasks of the SPA are:

- Registering the service with the VHE-M.
- Processing subscription/unsubscription of users to the service
- Negotiating subscription contracts.

- Performing service billing and eventually communicating with the VHE-M for the unified billing service.
- With the VHE-M, instructing and activating the service interface agent (SIA).
- With the SIA providing the service to the user.

Two types of SPAs may exist in the VHE system: SPAs for external services and SPAs for internal services. An SPA for internal service implements in addition to the tasks on top, the processing logic for the service. An SAP for external service doesn't implement the service's process, but only control the access to an external service. SPAs for external services allow to define intelligent interface that encompass non-intelligent services and integrate them to the system according to the VHE concepts.

4.1.5 User Profile Agent

The user profile agent (UPA) can be viewed as a generic agent from the home environment carrying information about a user to a remote terminal. The UPA is created only for remote terminals that have the ability to accommodate mobile software. Otherwise, the VHE manager performs the UPA's tasks. Hence, the UPA represent the user in an agent-based scenario. The main tasks of the UPA are:

- Providing the TA and SIA with the user information (profiles, identification).
- Managing and synchronising the profile replication and update with the VHE-M.
- Providing in co-operation with the TA the user with an interface for managing his profiles
- Providing the necessary information for authenticating the user.

The UPA is instructed and initiated by the VHE-M when contacted by a user for a VHE session.

4.2 Service Provisioning Scenarios

4.2.1 Starting the VHE Service

First the VHE-M must be started and bound to a certain name and address, so that any other agent or terminal can communicate with it. When started, the VHE-M becomes the entry point to the VHE system.

Three cases have to be distinguished depending on the terminal type used to contact the VHE.

1. If the terminal can't accommodate mobile software, the VHE-M starts a local TA in the VHE server. The created TA manages the communication according to the terminal type (e.g. provides a vocal interface or a GUI interface that fits the terminal features)
2. If the terminal can accommodate mobile software but there's no previous version of TA installed on it, the VHE-M initiate and send a terminal agent to the terminal for presenting the VHE interface to the user.
3. If the terminal has a terminal agent already installed on it, the VHE-M checks the version of the TA and eventually downloads the new version if it is the case.

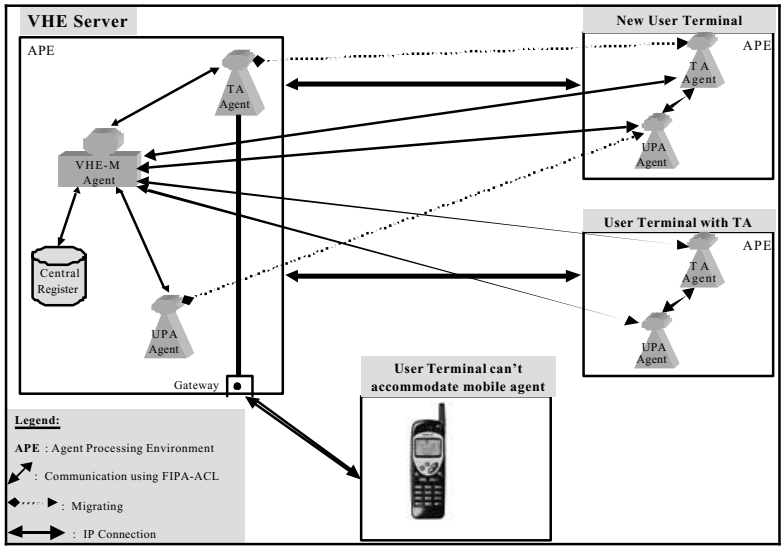


Fig. 2. Different scenarios for starting a VHE session

4.2.2 Service Registration

The specific service provider agent binds itself to the local namespace and sends a registration message to the VHE-M (1). The VHE-M stores this information (e.g. service name, IP address of the host the service is running, etc.) in the central register (2) and informs the SPA if the registration was successful or not (3).

A generic service interface agent is instantiated by the SPA and sent to the VHE server (4)&(5). This SIA copy is initialised an instructed (6) by the VHE and used for presenting the service to the user each time this service is requested.

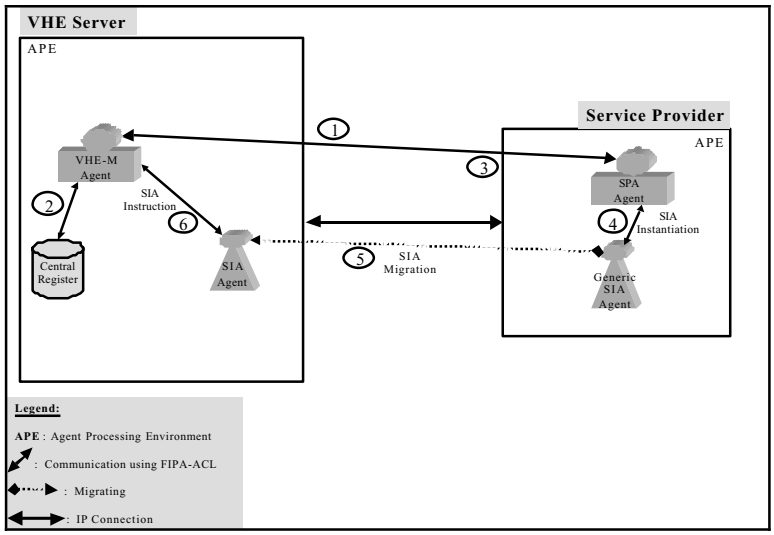


Fig. 3. Service registration scenario

4.2.3 User Authentication

The TA presents an interface to the user to enter his information (name, password, etc.), then an authentication message is built and sent to the VHE-M (1). The VHE-M, checks the user authentication data with the user profile stored in the central register (2). If the authentication is successful, it checks the current TA version with the TA version on the server. If a newer version is available it is automatically downloaded and installed on the terminal (3)&(4). Then the VHE-M initiates and sent a user profile agent (UPA) with the user profiles and a list of all available and subscribed services to the TA (5)&(6). If the authentication isn't successful a message is sent back to inform the user (7).

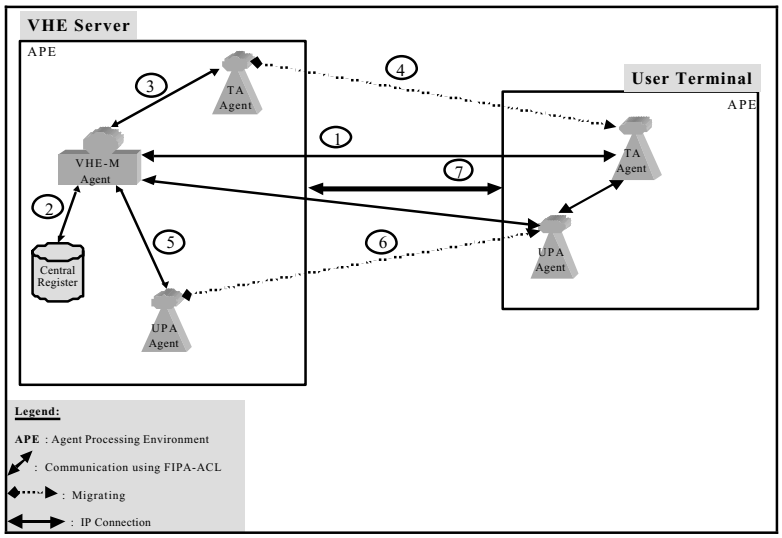


Fig. 4. User authentication scenario

4.2.4 Recovery from an Interrupted VHE Session

When a VHE session is accidentally interrupted all the closing process will not be performed. Hence, during the next VHE session opening the necessary update and synchronisation is done. After the user authentication and before presenting the VHE service to the user, the TA communicates the information about the old services using to the VHE-M (1)&(2) and the UPA performs the profile synchronisation with the VHE (3)&(2). Then, the VHE-M updates the TA and the UPA with the new data and logic if there's new versions. The VHE service is then presented to the user (4).

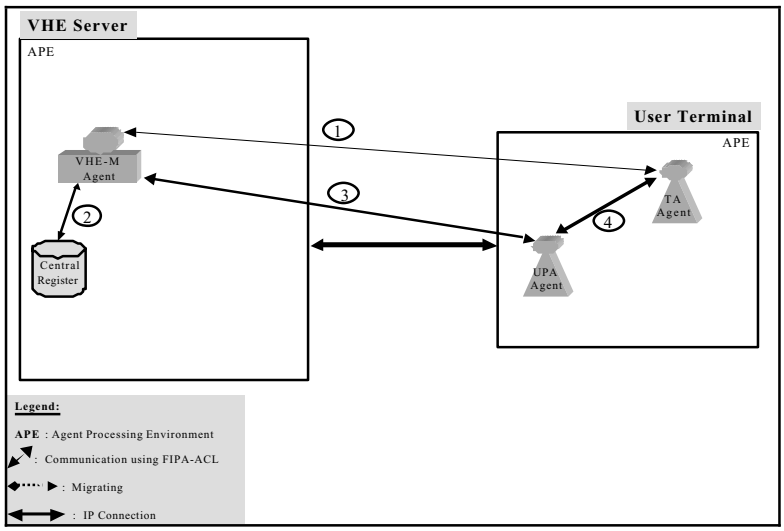


Fig. 5. Recovery from an interrupted VHE session

4.2.5 Service Request

After user authentication, all subscribed services are displayed on the terminal to the user. The user selects a service using the interface presented by the TA, then the TA build and sends a request message to the VHE-M. The VHE-M gets all relevant information from the central register and communicates with the SPA in order to instruct and initiate a service interface agent. The SIA moves to the terminal and presents the requested service by communicating with the TA and the UPA.

4.2.6 Using a Service

Starting a subscribed service, the TA sends the service request to the VHE-M to check whether the user is authorised to access the requested service (1). If so, the VHE-M in co-operation with the SPA instruct the SIA to transfer the service specific GUI to the terminal (2)&(3). The chosen GUI is adapted to the network and terminal features. The required data concerning the network, the terminal features and the user preferences are delivered by the UPA sent by the VHE-M at the beginning of the session (4).

After the SIA arrives to the terminal the connection with the VHE-M can be dropped and the connection with the SPA is set. Here it can be distinguished between two possibilities depending on the realisation of the requested service. First, the complete service provider agent can be cloned and moved to the terminal (5)&(6), this case is more suitable for services that process a real-time task to the user or process a huge database located in the terminal (or in the visited network). Second the SPA can't be moved, in this case a connection (RMI or Messaging) between the SIA and the SPA is required (7).

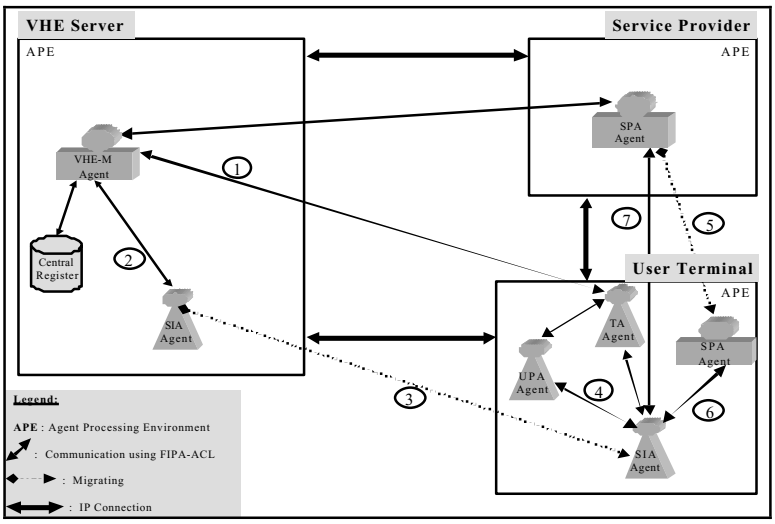


Fig. 6. Opening a service session

4.2.7 Closing a Service Session

After the user closes a service session, there's some housekeeping to do. First the SIA communicates the information about the service using to the TA and communicates the changes made by the user on his preferences to the UPA. A goodbye message is also communicated to the SPA.

Depending on the available resources in the terminal, the SIA and eventually the SPA (for cases where the SPA has moved to the terminal) will be deleted or stored using a caching mechanisms to keep the agent code for future service requests and just updates the data. The control is returned to the TA, which restart the VHE interface.

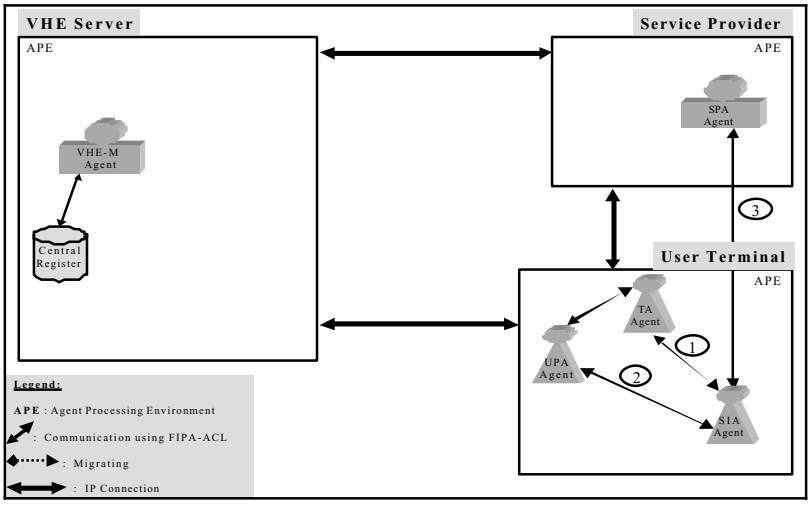


Fig. 7. Closing a service session

4.2.8 Closing a VHE Session

After the user chose to close a VHE session, there's some information to transmit to the VHE-M and some housekeeping to do. First the TA communicates the information

about the services using to the VHE-M (1)&(2) and the UPA performs the profile synchronisation with the VHE (3)&(2). A goodbye message is sent to the VHE-M (4). The connection is then closed and depending on the available resources in the terminal, the TA and the UPA will be deleted or stored using a caching mechanisms to keep the agents' code for future VHE sessions and just updates the data.

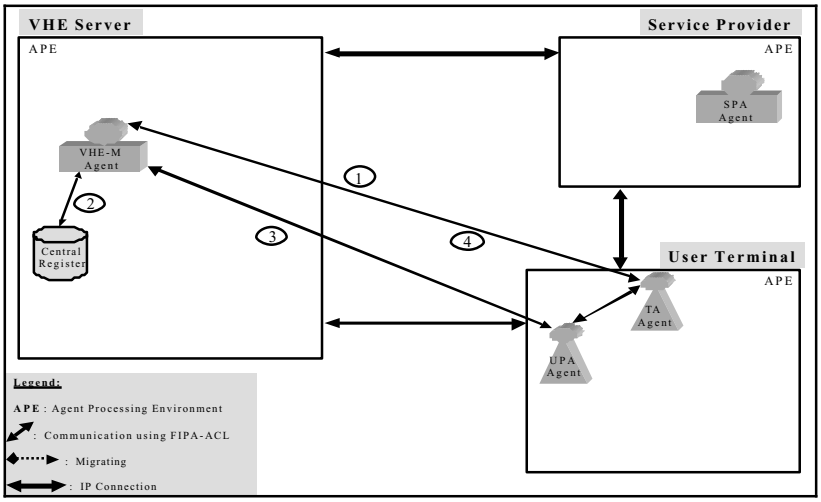


Fig. 8. Closing a VHE session

4.2.9 Service Subscription / Unsubscription

After user authentication, the TA presents all available services to the user. To subscribe to a new service, the user enters his bid specifications (e.g. price, QoS, time of use, etc.) and the TA build and sends a subscription message to the VHE-M. This later broadcast the message to all registered providers for this service to appeal them for their offers. The SPA's offers are sent back to the TA, which present them to the user.
If some offers are satisfactory the user made his choice and subscription contract is sent to the chosen provider and to the VHE-M. If none of the offers is satisfactory the user can choose between starting a negotiation session with a specific provider (using a propose/counter-propose model), making an other request by modifying his specification or cancelling his subscription request (figure 9).
When a contract is set-up for a new service the VHE-M and the SPA of the service communicate for setting the service profile which is added to the user profiles stored on the central register. Also a new version of the TA is downloaded to the terminal.
To unsubscribe to a service, the user selects the service from his subscribed services and the TA build and sends an unsubscription message to the VHE-M. The VHE-M informs the SPA, updates the user profiles and downloads a new version of the TA and UPA to the terminal if it is still connected.

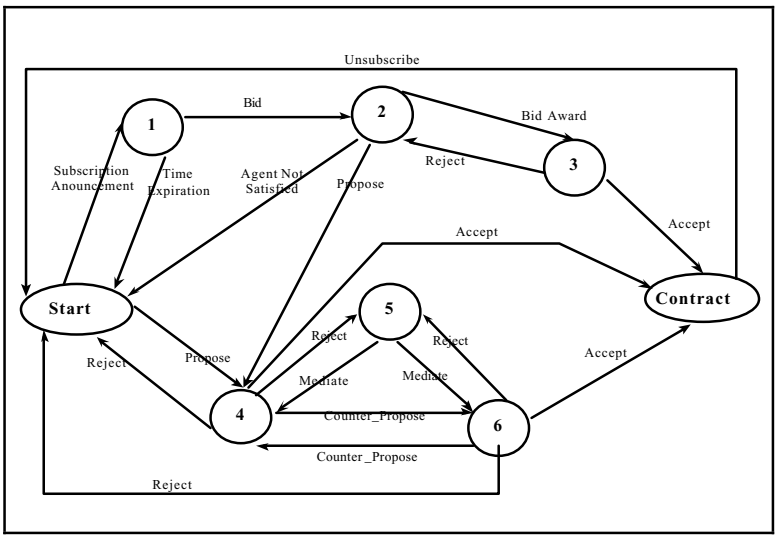


Fig. 9. Diagram state for elaborating a subscription contract

4.2.10 Switching between Different Types of Terminals

In this scenario the user may start using a service from an intelligent terminal that can accommodate mobile software (e.g. PDA, Laptop), but during the session he decides to switch to an other terminal that don't have the ability to accommodate mobile software (e.g. PSTN Phone) and vice versa. Figures 10 and figure 11 show the two steps of the scenario.

Step1: the user start with an intelligent terminal (Figure10):

The service interface agent (SIA) migrates from the VHE server to the intelligent terminal, presents the service interface to the user and manages the connection session between the service provider (SAP) and the terminal. The communication between the VHE-M and the SPA serves only for instructing and initiating the SIA. So, at this time the connection between the VHE server and the SPA can be dropped. The inactive PSTN connections are represented in the figures using grey lines with an "X" over them. The TA agent communicate with the SIA and the VHE-M in order to manage the service interaction (e.g. receiving notifications from other services)

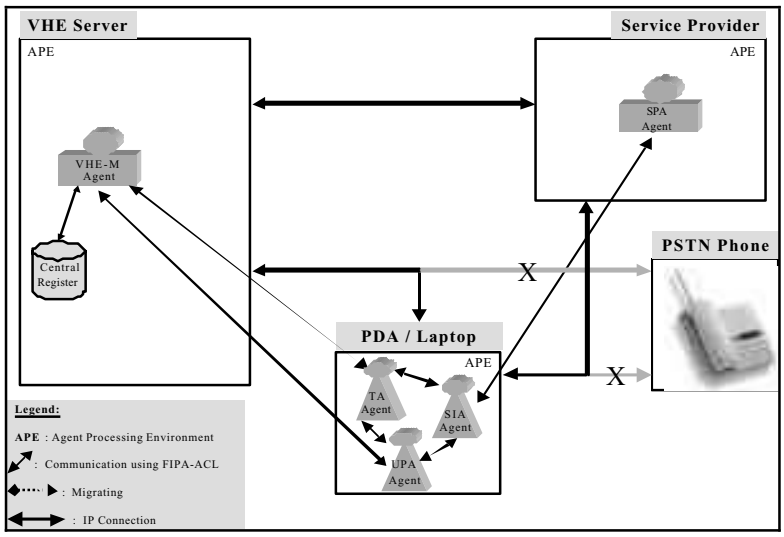


Fig. 10. Service session using intelligent terminal

Step2: the user switch to a non-intelligent terminal (Figure11):

When the user decides to switch to an other terminal. After entering this request, the TA communicates the request and the state of the current session to the VHE-M. The VHE-M starts a new PSTN TA and SIA, and instructs them with the session's state got from the intelligent terminal. The new TA and SIA will reside in the VHE server and manage the new session by communicating with the SPA. In this case, the connection between the VHE server and the SPA server must remain active during the service session. The session with the PDA/Laptop may be disconnected or only suspended depending if the PDA/Laptop and PSTN Phone use the same connection line or different lines.

Other solution that can be considered consists of moving the new TA and SIA to the SPA server. This solution appears to be more optimal than the one we have considered on top because only the necessary servers and connections needed for performing the service remain active during the service session (SPA server and the visited network server). However, this solution supposes two requirements: (1) the SPA server can accommodate mobile agents and (2) the SPA server accepts to allocate resources for the migrating agents.

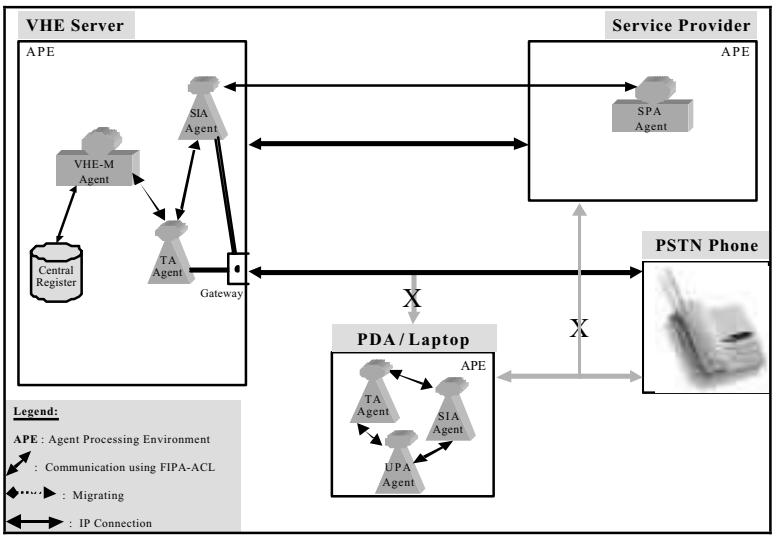


Fig. 11. Service session PSTN terminal

The advantages of keeping all time the TA and SIA in the service control loop are:

- Providing the user with an online services interaction (i.e. receiving notifications from other services even if he switch to an other terminal)
- Providing the user with the same "look and feel" without need of configuration for new terminals.
- All modifications that the user made on his profiles or his service rules from one terminal are reported automatically to the VHE-M and still be available even if he roam or switch from one terminal to an other.
- Providing the user with a coherent integration of service
- Providing the user with an extensible framework that can accommodate new services and assure their interaction with existing ones.

5 Conclusion and Future Works

The provision of personalised telecommunication services to subscribers in a global networking environment is a complex problem that is intensified when those subscribers are mobile and wish to access the services from many different terminals. In this paper we have showed that the combination of the VHE concept and agent technology enhance services interaction and allow easy and rapid deployment of new sophisticated Tele-services. In particular, the architecture described shows that the "look and feel" of services and the "on demand provisioning" of services can be easily realised with the help of agent technology.
We are currently at the design stage of this work. Hence, our next step is to implement and evaluate the system.

References

1. UMTS Forum. UMTS/IMT2000 spectrum. Report No: 6. December 1998.
2. CAMELEON. Project AC341. Communication Agents for Enhancements in a Logical Environment of Open Networks. Delivrable 09. November, 1999. Web: http://www.comnets.rwth-aachen.de/~cameleon/cameleon/D09_final.zip
3. N. R. Jennings. A roadmap of agent research and development. International journal of autonomous agents and multi-agent systems. Vol1, No1, pp:7-38. 1998.
4. 3rd Generation Partnership Project (3GPP). Technical Specification. The Virtual Home Environment (3G TS 22.121, V.3.0.0). June 1999.
5. D. Milojicic, F. Douglis and R. Wheeler. Mobility: Processes, Computers and Agents. Publisher Addison Wesley. February 1999.
6. S. Brown. Implementing Virtual Private Networks (VPNs). Publisher, Mc Graw Hill. May 1999.
7. A. Fuggetta, G. P. Picco and G. Vigna. Understanding code mobility. IEEE Transactions on software engineering, Vol:24, No:5, pp:342-361. May 1998.
8. A. Birrel and B.J. Nelson. Implementing Remote Procedure Calls. ACM Transactions on computer systems, pp:39-59. February 1984.

Handling Subscription in a Mobile Agent Based Service Environment for Internet Telephony: Swapping Agents

Roch H. Glitho[1], Bertrand E. Lenou[1,2] and Samuel Pierre[2]

[1]Ericsson Research Canada
[2]Ecole Polytechnique, Montreal, Canada

ABSTRACT

Internet telephony brings a host of opportunities. Cost can be reduced and new services can be engineered. Mobile agent based service architectures for Internet telephony, have emerged in the recent past. They stipulate the use of mobile agents that act as folders and carry the executables of services (or pointers to the executables). Carrying executables (or pointers to executables) in a mobile agent brings new challenges. Subscription handling is among them. This paper tackles the subscription problem. There are two possible approaches for upgrading a mobile agent that carries services, when the user subscribes to new service(s): agent swapping and on-the-fly updating. In the first approach, the agent that carries the services is swapped with a new agent that carries both the old and the new services. In the second approach, the new services are inserted in the agent on the fly. In this paper, we give an overview of our mobile agent based service architecture, state the subscription problem, derive requirements, and propose tentative solutions based on the swapping approach. The requirements include minimization of the duration of service interruption, scalability, and programming simplicity. The problem although related to software replacement, is significantly different. The two solutions proposed in the paper, *smooth swapping and abrupt swapping*, scale to a large extent, as shown by the prototyping results. There is no service interruption with the smooth swapping. Unfortunately, it requires that the old and the new agents co-exist on the user device or network node for a short while. It might therefore not be applicable to small footprint devices. The abrupt swapping does not require the co-existence of the old and the new agents and is

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 49-66, 2000.

therefore applicable to all types of devices. Service is however interrupted. Fortunately, the duration is insignificant as shown by the prototyping results. The bad news is that service programming is relatively complex in both cases.

1. Introduction

Mobile agents have emerged in the mid-1990s and have been used in fields ranging from network management to information retrieval, and including service engineering. Several tutorials [1,2] have been published on the topic. Services are differentiating factors and keys to service providers' survival and success. Some examples are call diversion, toll free calls, split-charging, and stock quotes.

Research in the use of mobile agents for service engineering is no novelty. It dates back to the early days of mobile agents. Most of the activities in the field [3,4], are however still confined to circuit switched networks. The key issues tackled by the mobile agent based service architectures for circuit switched telephony are related to the centralized nature of the intelligent network (IN) architecture, the service engineering paradigm traditionally used in the telecommunications world.

The proposed architectures stipulate the implementation of service logic programs (SLPs) as mobile agents. These agents can move to appropriate locations in the network, depending on the load. In these architectures, there is generally a mobile agent for each service to which the user has subscribed.

Two sets of standards are now emerging for Internet telephony: H.323 from the ITU-T and SIP from the IETF. They include service architecture related specifications. The weaknesses of the proposed architectures are now well known [5]. While these architectures are constantly evolving, alternatives are emerging.

Mobile agent based service architectures are among the alternatives to today's IETF and ITU-T specifications. The emerging mobile agent based service architectures for Internet telephony [6,7,8,9] depart from their circuit switched counterparts approach of having one mobile agent per service. They propose mobile agents that act as folders and carry the services including logic and data (actual executables or pointers to the executables, actual data or pointers to data).

Carrying services in a mobile agent raises new questions. Subscription handling is among them because the agent needs to be upgraded when the user subscribes to new services. There are two possible approaches for

upgrading an agent that carries services: agent swapping and on-the-fly updating. In the first approach, the agent is swapped with a new agent that carries both the old and the new services. In the second approach, the new services are inserted in the agent on the fly.

This paper focuses on the first approach. In the next section, we give an overview of our mobile agent based service architecture for Internet telephony. We then state the problem, derive a comprehensive set of requirements, and examine related work, namely software replacement. The section after that proposes two tentative solutions: *smooth swapping and abrupt swapping*. After that we describe the prototype and evaluate the proposed solutions in the light of the requirements and the prototyping results. We finally conclude and identify the items for future work.

2. Our Mobile Agent Based Service Architecture

Earlier versions of the architecture have been described in previous work [6,7,8]. The main entities are the mobile service agent (MSA), the service management unit (SMU), and the service creation unit (SCU). The architecture is depicted below along with the service provisioning process. MSA, SMU and SCU are subsequently described.

Service Provisioning Process

1. Programmers develop services using the Service Creation Environment (SCE). The SCE right now is made of a small API and a programming guideline to develop services. The programming guideline specifies the construction of each service as a self-contained entity.
2. Services are stored in the Service Implementation Repository.
3. SCU informs the SMU of the existence of new services.
4. SMU forwards new services names to Service Publication Unit (SPU).
5. Users subscribe to new services through the SPU.
6. The SPU sends the subscription parameters to the SMU.
7. The SMU sends MSAs (one for each group of services that could operate together) containing the services or service pointers to either the user Gatekeeper / proxy or the user device, depending on the MSAs.
8. MSAs download the services from the SCU. They later execute, co-ordinate and enable customization of the services.

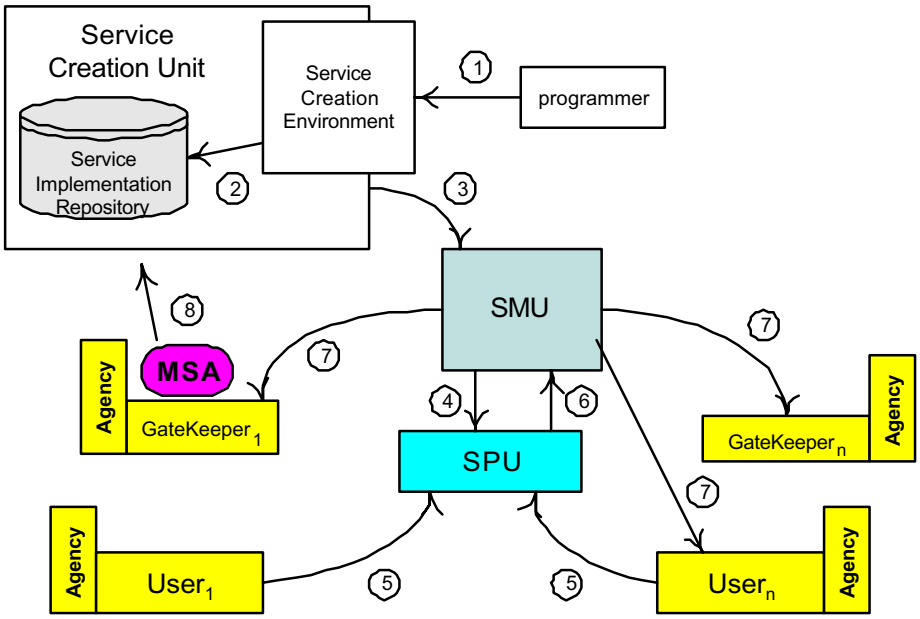


Figure 1- Our mobile agent based service architecture for Internet Telephony

2.1. The Mobile Service Agent

The MSA is the key entity. It acts as a folder, and carries for each service belonging to a subset of services to which the user has subscribed:

- The logic (executable code or pointer to the executable code)
- The data (actual data or pointer to the data)

Any and every given service for which the user has a subscription is carried by one and only one MSA. In any given instantiation of this architecture, the number of MSA per subscriber is fixed. There are two extreme cases:

- The number of MSA per subscriber is equal to the number of services to which he has subscribed. This corresponds to the case of the mobile agent based service architectures for circuit switched telephony [3,4] meaning that each service is implemented as mobile agent.
- There is 1 MSA per subscriber. This MSA carries all the services to which the user has subscribed. This is the case presented in reference [6].

The MSA has a brain that allows it to perform a well-defined set of operations. An example of operation is the possibility to start the execution of a service it carries. The MSA is depicted below.

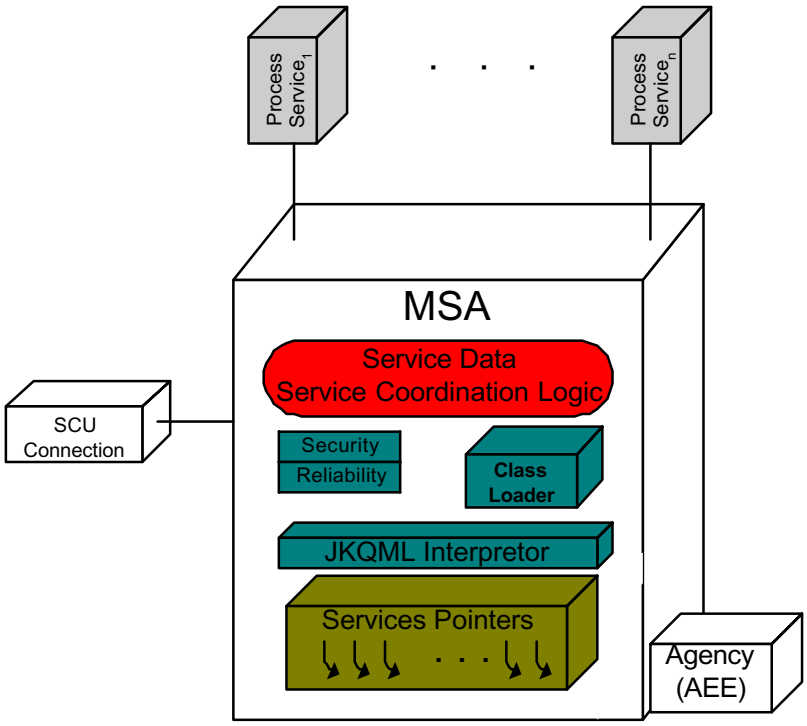


Figure 2 – The mobile service agent

2.2. The Service Creation Unit and the Service Management Unit

There are two types of service units in the architecture, the service creation unit (SCU) and the service management unit:

- The (SCU) allow the creation of the services to which users could subscribe. There are two key functional entities in an SCU. The first is the service creation environment. It includes the tools used for creating the services that are made available to users. It may include a mobile agent development environment in the case where services are implemented as mobile agents. The second is the service repository. It contains the executable codes of all services created in the unit. MSAs carry these very executable codes (or pointers to entries in these repositories).

- The service management unit (SMU) manages subscriber and service life cycles. Its functionality includes:
  - Publication of the list of available services. This list reflects all services to which users can subscribe at any given time. It is updated every time a new service is created in a SCU.

- MSA creation. The MSA is created the first time the user subscribes to service(s) belonging to the category of services carried by that type of MSA. The SMU creates an MSA by assembling MSA brain, and services (or proxies).
- MSA management. Besides updating, MSA management includes moving the MSA to the appropriate node (or terminal) during user's intra domain and inter domain roaming.
- User profile creation and management. The profile is created the first time the user subscribes to service(s). It contains the list of MSAs that carry the services to which the user has subscribed, and eventually the list of services.

3. The Subscription Problem

We first state the problem and derive a comprehensive set of requirements. We then give a very brief overview of the literature on online software replacement and show that replacing software (as per previous art) is significantly different from swapping agents.

3.1. Problem Statement and Requirements

Let us assume that an MSA has already been created for a given user, and includes A, B and C, services to which the user has already subscribed. To be more specific, the MSA will include the logic (executable code or pointer to executable code), plus the data (or pointer to the data) of each of the three services and will be aware that it contains them. It is important to note that the data it contains (or it points to) can be customized by the user.

Let us now assume that the user decides to subscribe to an additional service D. An interesting question is how the MSA is upgraded in order to include D in addition to A, B, and C. Intuitively, we can think of two approaches. The first consists of swapping the MSA including A, B, C with a new MSA that includes A, B, C and D, and is aware that it contains them. The second consists of updating the MSA on the fly in order to include D to the services it contains, and to make it aware that it now contains D in addition to A,B, and C.

These two approaches have pros and cons. A set of preliminary requirements has been derived to analyze the two approaches, in order to find a solution that is optimal with respect to these very requirements. The requirements are listed below.

1) Service interruption due to MSA upgrading should be minimal.
2) Time lag between a request for subscription and actual availability for use should be minimal.
3) Upgrading should not impact the behavior of existing services (i.e. user customization should not be lost).
4) The solution should be simple to implement.
5) The solution should scale in terms of the total number of services (i.e. services for which the user already has subscribed plus services for which the user has launched a subscription request).
6) The solution should be mobile agent platform independent. There are several platforms on the market. The solution should not use the specifics of none.

Requirements 1), 3) and 5) are the most difficult to meet in the case of the agent swapping strategy. Service is likely to be interrupted while the old MSA is being swapped with the new one. Furthermore solutions may not scale in terms of the total number of services, because a new MSA needs to be built from scratch. Time lag between request for subscription and availability may become prohibitive when the number of services increases, assuming that the new agent needs to reload the code of the services to which the user has already subscribed. Furthermore data customized by users should be kept.

3.2. Related Work

There is substantial literature on online software replacement. Reference [10] although a bit dated gives a good overview of the issues and the solutions, while references [11,16] give a more updated view of the various techniques.

The primary concern of on-line software replacement is to replace or update a program version without service interruption. Most of the related literature deals with dynamic updating and the inherent difficulty to preserve program correctness (type safety, name binding, etc.) after the updates having been done.

Dynamic updating is concerned with adding code to a running application, without halting the application. With agent swapping, we are trying to change the whole application – the old MSA and its service threads – with a new MSA that contains the previous services (same or newer versions) and some new services. The challenge is to perform the swapping while minimizing downtime and fulfilling all the other requirements already listed in the paper.

In on-line software updating, user customized data is preserved as long as that solution can guarantee program correctness after the updates. The

software has however to be designed using a specific framework. On the other hand, the principal difficulty with swapping is to keep user customized data. Furthermore the constraint of using a specific framework for the design of the MSA is not acceptable since it contains a wide variety of services.

4. Smooth Swapping and Abrupt Swapping

This section proposes two tentative solutions to the problem of agent swapping: *smooth swapping and abrupt swapping*. We start by sketching them. We then describe them in detail in terms of the operations they require the MSA to perform. The basic operations performed by an MSA are described first. The operation of exchange of customized data is then described, followed by the operation of synchronization.

4.1. Sketches

In both smooth and abrupt swapping, the SMU starts by assembling a new MSA that contains pointers to the executables of both the old and the new services. The MSA then downloads the services.

- In case of smooth swapping, the new MSA moves to the site where the old MSA resides, after having downloaded the executables. It then gets the data customized by the user (if any) from the old MSA, then becomes active. After having sent the customized data to the new MSA, the old MSA becomes inactive. However, if services are being executed, the execution is not interrupted. It is completed before the old MSA becomes inactive.

- In case of abrupt swapping, the new MSA does not move to the site where the old MSA resides, after having downloaded the executables. It first gets the data customized by the user (if any) from the old MSA. In the first alternative of this solution, the old MSA stops the services being executed (if any), then becomes inactive. In the second alternative, the execution is completed, before the old MSA becomes inactive. The new MSA moves to the site where the old MSA resides as soon as the old MSA is inactive. It then becomes active and restarts the service executions that have been interrupted by the old MSA (if any).

Smooth swapping requires that old and new MSA co-exist on the same site (e.g. user device) while swapping occurs. This is however not always possible, especially with small footprint devices. This is the main driving

force behind abrupt swapping. While it is theoretically possible to wait till the services being executed finish before initiating the swapping, we did not contemplate this, because the execution of some services might take quite a while. An example is a stock quote service initiated by the user early in the morning and which should report every hour the prices of selected shares, till the stock market closes.

These two sketches raise several issues:

- How to transfer user customized data between MSAs and still put minimal constraints on the MSA and its services (interfaces, programming model, etc.).
- How can to maintain consistency between two running versions of the same service, in the case of smooth swapping.

4.2. Service Downloading and Initiation

The brain of an MSA can perform two basic operations:

- Download a service
- Start the execution of a previously downloaded service

This done through the custom class loader: "MSAClassLoader". MSAClassLoader subclass Java class loader. Figure 3.1 presents part of Java class loader. Figure 3.2 depicts the MSA class loader.

```
public abstract class ClassLoader
{
        public Class loadClass(String name);
        protected Class findClass(String name);
        .
        .
        .
}
```

Figure 3.1: JDK class loader

```
class MSAClassLoader extends ClassLoader
{
        String scu_location;
        public MSAClassLoader (String location) {
                scu_location = location;
        }
        protected Class findClass(String name){
                byte[] classbytes = getClassCode(name);

        return(defineClass(name,classbytes,0,classbytes.length));
        }
        public Class loadClass(String name) {…}
        byte[] getClassCode(String name) {…}
        byte[] getClassFromArchive(String name) {…}
        .
        .
}
```

Figure 3.2: MSA class loader

Applications can define multiple class loaders. Those class loaders can then be used to remotely download and start any program. That's what the MSA does as shown by figure 3.3. Services are typically bundled in an Java archive file (.jar)

```
class MSAServiceProcess extends Thread
{
   String service;
   Object Data; String protocol;
  public MSAServiceProcess (ThreadGroup th, String service, Object
sdata) {
        super(th,service);
        Data = sdata;
}
void addServiceClasses(String className){…}
  Object getServiceParam(…){…}
void run() {
        getServiceRules();
        if(condition 1){
                launchService(msa, service1)
        }
        else{
        …
        }
}
  void launchService(MSA msa, String mainclass){
        MSAClassLoader ms = new MSAClassLoader(loc);
        Class c = ms.loadClass(mainclass);
```

```
            Class[] carray = c.getInterfaces();
            If (carray[I].getName != "IMSAService"){…}
            Object serv = c.newInstance();
            Method m = serv.getClass().getMethod("main", new Class[]
{…})
            Object o1 = getServiceParam(…);
            m.invoke(null, new Object[] {o1});
            try{
                        sleep();
            }catch(InterruptedException e){
                ….
            }
        }
            .
            .
            .
    }
```

Figure 3.3: MSA launching a service

4.3. Exchange of Customized Data

The behavior and the invocation of the services contained in a MSA might be customized. MSAs allow users to specify the execution parameters of their services. MSAs offer a uniform interface subscribers use to launch services. Service parameters entered through that interface will be saved on user request. Subsequent instances of that service can be started automatically with those parameters if the user wishes so. One of the services we used to test the MSA was a meeting planner (we describe it later on in section 4.1). Here, subscribers had the option of entering a set of participants, potential meeting dates (days) and scheduled execution dates. On the scheduled execution date, the MSA launches the meeting planner that goes on and tries to find an appropriate slot (hour) for the next meeting.

Users can also edit rules that will be followed before a service is launched. The simplest example of a rule is given in the preceding paragraph where the user specifies a regular or an occasional execution date for one or many services. In our telephony MSAs, users entered rules that governed the call processing logic of the MSA. During our prototyping activities, the telephony MSA contained services like Call Forward, Voice Messaging and e-mail notification. At execution, the MSA popped up a GUI that testers used to easily indicate which of the services should take precedence and handle the call. Cases where rules could be used arise mostly in multimedia (example) and office (example) applications.

The issue here is to preserve the service parameters and the rules that apply to service invocation while swapping MSAs. Services data can be saved in an URL. The new MSA then passes that data to the new MSA through method calls. The new MSA uses the name server to obtain a reference to the old MSA and vice-versa. Alternatively, MSAs exchange services data and execution logic in KQML messages. We used Java KQML (JKQML) to test this approach. Both alternatives are viable. We however, prefer JKQML because the first approach implies that MSAs must implement a predefined interface that other MSAs will use to transmit service information. We don't want to impose interfaces on MSAs when it can be avoided. With the service ontology developed during prototyping, any pair of MSA can converse to exchange service information. No specific set of functions is required. This fits in the agent development paradigm.

Conversely, there is no requirement on any service to implement any interface. There is only a programming notion that services be self-contained. That is they shouldn't hold references to other services or try to invoke other services. That requirement partially solves the features interaction problem that is acute in IN [17]. JavaBeans might are a solution in that direction. However, we do not use most of the bean features at this stage. Consequently, it is unnecessary to impose it as model. Services thus become atomic units and the interaction complexity is all hidden inside the MSA in the Service Co-ordination Logic module. Our prototype so far dealt with pretty straightforward services. Coordinating could however be more complicated with a bigger set of services.

4.4. Synchronization

Here, we are concerned with synchronizing the states of the MSA during swapping. Synchronizing the states is essentially making sure that user customized data and user-defined rules are transferred from the old MSA to the new MSA. Service synchronization roughly takes place like shown below.

```
Class MSA {
…
private ThreadGroup th = new ThreadGroup();
public void startService()
{
   KQML request = new KQML();
   request.setPerformativ e(KQML.TELL);  //set KQML performatives
   …            // to tell the other agent to send a service customization
   ServerSocket sock = new ServerSocket // Open socket to receive service
   sock.accept();  // data.  Uses HTTP as protocol here

   request.setOntology(MSA.ontology);
```

```
    request.setContent("get rules" + service);
    saveRules();
    If(smooth_swapping){
      MSAServiceProcess mp = new MSAServiceProcess(th, service, data);
      mp.start();
    }
    if(abrupt_swapping) {
      request.setContent("Stop your services");
      …
    }
  }
  …
  // some other methods
  }
```

Figure 3.5: Strategies of agent swapping

As shown in the figure, if the smooth swapping strategy is used, user service invocations will result in new service threads being started concurrently to the ones running in the old MSA. With the other strategy, the new MSA asks the old MSA to stop that particular service thread.

5. Prototyping and Evaluation

Our prototyping activities involved a SMU, a SCU and several MSAs. We tested agent swapping with several mobile, still code based, and mobile agent based services. The first mobile agent-based service is a meeting planner. The meeting planner dispatches a mobile agent to several devices; that agent tries to get a consensus date and hour (from the attendants) to hold a meeting.

The second mobile agent-based service tries to find the shortest path to route a call. It sends several cooperative agents on the network, with each of the agents containing a network map. Next, the agents converse, divide the task among them, and then each goes on its exploratory path. Once one reaches the destination, it informs the others who retreat back home.

The MSA was implemented in Java, on top of the Voyager mobile agent platform. We are currently porting it to Grasshopper. We first present our results. Next, we present some unresolved issues with our implementation.

The two agent swapping strategies are illustrated side by side in figure 3.6 and figure 3.7. In figure 3.6, objects are recreated and the old MSA is not touched. It will be garbage-collected once its services being executed terminate. In the second solution, second alternative (figure 3.7), services are stopped and recreated in the new MSA.

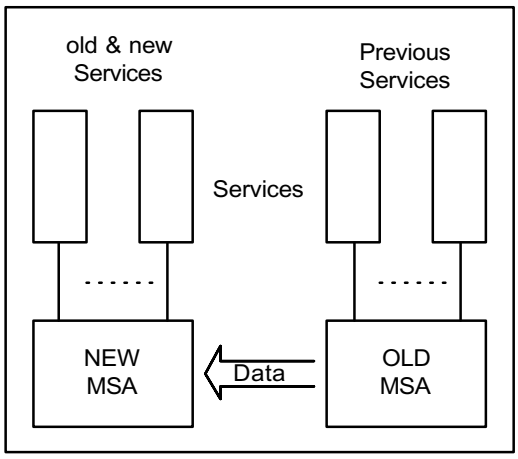


Figure 3.6: Smooth Swapping

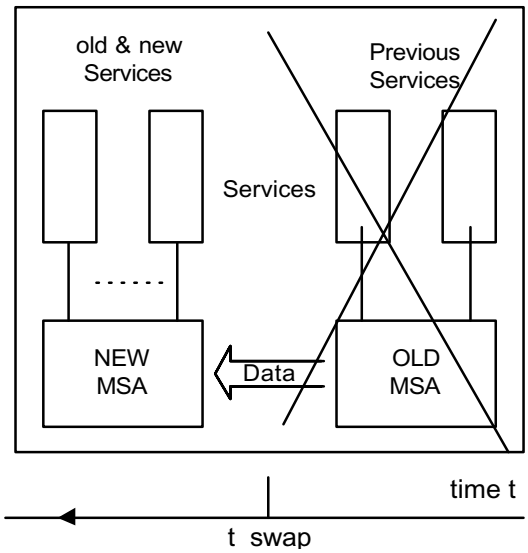


Figure 3.7: Abrupt Swapping

We measured the quantifiable requirements of our solution namely: service interruption duration, time lag for availability of services and scalability. Our testbed consisted of a SMU located on a 400 MHz Intel Pentium Pro running Solaris 2.5.6; a SCU located on a 366 MHz Intel Pentium Pro running Windows NT 4.0 and 8 other computers (used for subscriptions) connected to Ericsson LAN.

We try to factor network latency into our results. Effectively, we run our tests during the peak office hours between 10 a.m. and 2 p.m. Ericsson LAN typically serves around 1500 employees on a unique hub during that period (10 to 2). The tests were made over a week; we compute the seven days average value for each of our series of tests.

The first series of tests involved twenty-one services with individual sizes varying from 500K to 10000K, with an increment of 500K. The services have also been assembled differently. Some were made up of just one file while others were made of up to 50 files.

The first conclusion of this first series was that the number of files that make a service -and accordingly the number of files to swap- only marginally affect the subscription delay. Rather, it is the service size that is the determining factor. This points out that I/O at the SCU (fetching and serving files) isn't the limiting factor. It is instead the network latency.

Figure 3.8 shows the time lag in seconds as function of the total size of all services (old and new) carried by the MSA, independently of how many

services are put together to reach the total size. As it is apparent from the results, the delay is small up to a total size of 10000K.

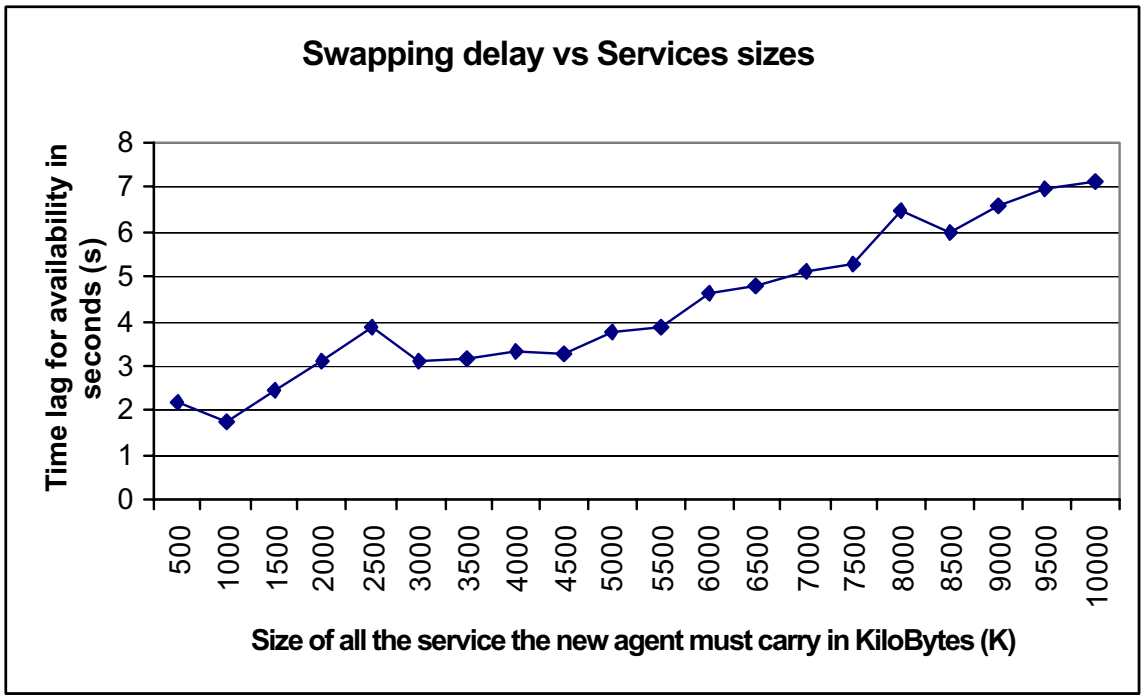


Figure 3.8: MSA swapping time lag evolution

In the second set of runs, we took arbitrarily big services and subscribed to an arbitrarily big number of services (up to 150). The results are illustrated in figure 3.9. Performance falls steeply when the total size of the service carried by the MSA reaches 30 Gigabytes.

We can safely say that a 30 G limit is more than enough for the current applications. WE can therefore conclude that the solution scales, that there is a short time lag for availability, and also that service interruption is indiscernible. We sum up those characteristics in table 3.1 below.

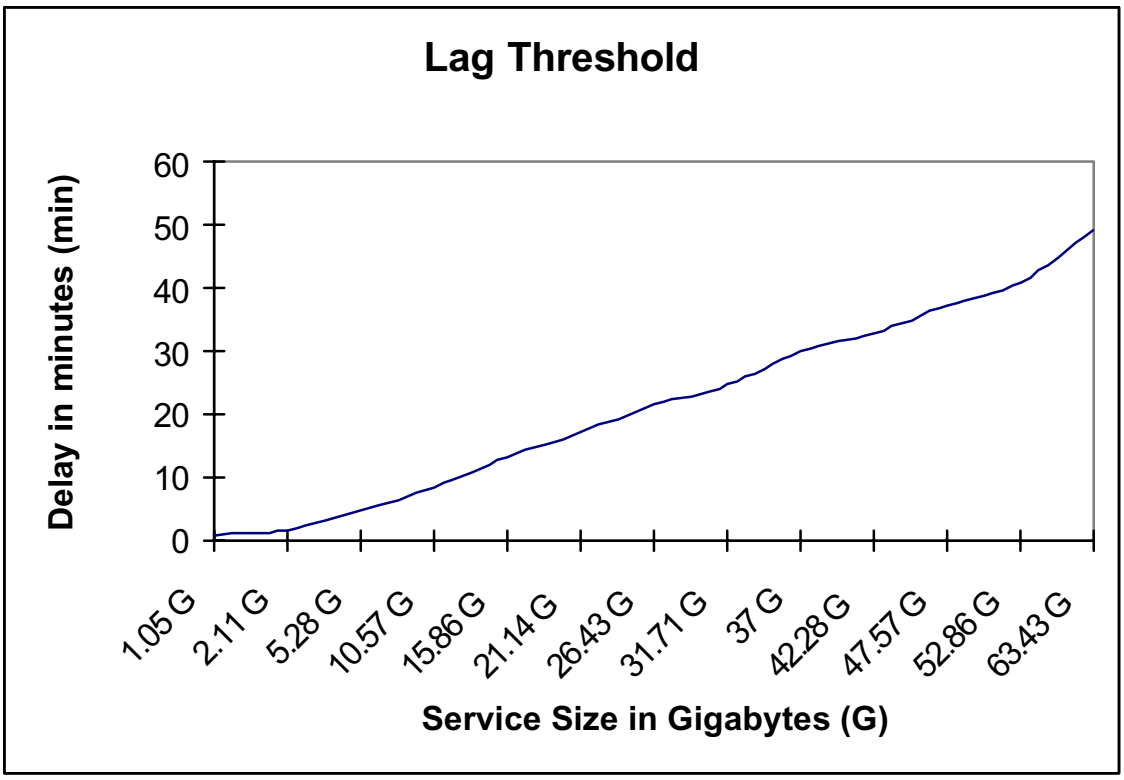


Figure 3.9 Service size / Lag Time threshold

5. Conclusion and Future Work

Handling subscription using swapping is one of the many possible solutions. The next logical alternative is on-the-fly upgrading. We've been working for a while on that approach. Unfortunately, like Malabarba *et al.* [13] founded out before us, there are several untraceable bugs that emerge when one tries to implement dynamic updating at the library level. Still, we have no choice but to redesign and try again, since modifying the Java Virtual Machine to support our MSA would take away the generality of our approach and render it moot.

Other approaches to on-the-fly updating (Oreizy and al[14]) impose specific architectural models and thus restricting the range of applications that would be hosted by the MSA. Right now, we are revisiting our solution and formalizing our conditions for validity of change. The work done by Gupta and al[16] with their theoretical framework might help. The granularity of our change – an entire service process – reduces the workload since we do not have to worry about method redirection.

	Impact on existing services	Service Interruption	Scalability
Abrupt Swapping	None	Yes, indiscernible in practice	Yes
Smooth Swapping	Customization made on the services in the old MSA during the swap is lost	None	Yes

	Time Lag for Availability	Simplicity	Platform Independence
Abrupt Swapping	Small, typically under 2 seconds	No	Yes
Smooth Swapping	Small, typically under 2 seconds	No	Yes

Table 3.1: Swapping Strategies Evaluation

References

1 A. Karmouch and V. A. Pham, Mobile Software Agents: An Overview, IEEE Communications Magazine, July 1998, Vol.36, No7

2 M. Perdikeas et al., Mobile Agents and Available Platforms, Computer Networks, Vol.31, No19, August 1999, pp. 1999-2016

3 M. Breugst and T. Magedanz, Mobile Agents - Enabling Technology for Active Networks Implementation, IEEE Network, May/June 1998, Vol. 12 No3, pp - 53-60

4 F. G. Chatzipapadopoulos, M. Perdikeas and I. Venieris, Mobiel Agent and CORBA Technologies in the Broadband Intelligent Network, IEEE Communications Magazine, June 2000

5 R. Glitho, Advanced Service Architectures for Internet Telephony: A Critical Overview, July/August 2000

6 J. Tang, T. White, B. Pagurek and R. Glitho, Advanced Service Architecture for H.323 Internet Protocol Telephony, Computer Communications, Vol. 23, No8, pp. 740-754

7 B. Pagurek, J. Tang, T. White and R. Glitho, Management of advanced services in H.323 Internet Protocol Telephony, IEEE Infocom 2000

8 R. Glitho and A. Wang, A Mobile Agent Based Service Architecture for Internet Telephony, ISS2000

9 M. Fisher, W. Chang and Th. Magedanz, Distributed IN services for mobile agent based Internet, IEEE IN Workshop 2000

10 M. Segal, O. Frieder, On-The-Fly Program modification: Systems for Dynamic Updating, IEEE Software, March 1993

11 S. Liang and G. Bracha, Dynamic Class Loading in the Java™ Virtual Machine, OOPSLA 1998

12 G. Hjálmtýsson and R. Gray, Dynamic C++ Classes, A lightweight mechanism to update code in a running program. In Proceedings of the USENIX Annual Technical Conference, New Orleans, Louisiana, June 1998. USENIX

13 S. Malabarba, R. Pandey, J. Gragg, E. Barr and J.F. Barnes, Runtime Support for Type-Safe Dynamic Java Classes, In Proceedings of the 14th European Conference on Object-Oriented Programming, 2000.

14 P. Oreizy, N. Medvidovic and R. Taylor, Architecture-Based Runtime Software Evolution, In Proceedings of the International Conference on Software Engineering, 1998.

15 Michael Franz, Dynamic Linking Of Software Components, IEEE Computer, March 1997

16 D. Gupta, P. Jalote, G. Barua, A Formal Framework for On-line Software Version Change, IEEE Transactions on Software engineering, Vol 22, No 2, February 1996

17 Y-J. Lin and N.D. Griffeth, Extending Telecommunication Systems: The Feature Interaction Problem, Computer, vol. 26, no 8, pp. 14-18, Aug 1993

Mobile Network Domain Agency for Managing Network Resources

Farag Sallabi, Ahmed Karmouch

Multimedia Information and Mobile Agents Research Laboratory
School of Information Technology and Engineering, University of Ottawa
E-mail: sallabi@site.uottawa.ca
karmouch@site.uottawa.ca

Abstract. For the increasing demand of QoS guarantee by recent multimedia applications, network protocols developers are stressing the routers by suggesting new services, which overload routers and affect their performance. Those new services include resource reservations in the network elements and QoS routing. The resource reservation services offer immediate and advance reservations that are handled by network elements. Resource reservations result in reservations states for admitted flows, which will not scale well for the Internet. In this work we present architecture for domain agency, which handles the resource reservations and the QoS routing in the domain. The domain agency is ***mobile***, where it exists in a network domain and whenever the domain is split into smaller domains, the parent domain agency dispatches child agencies to each new domain. The main components of the domain agency are *Domain Resource Reservation Agent*, *Quality of Service Routing Agent*, *Domain Admission Control Agent* and *Reservations State Agent*.

1 Introduction

Recent multimedia applications require stringent values of QoS. Introducing high quality video and audio signals have made it difficult for the existing infrastructure of the Internet to cope with those multimedia applications. Furthermore, recent applications are associated with user interactions, and the ability to browse different scenarios at the same time. In fact, these services made the researchers look for other solutions and even make changes to the existing Internet Infrastructure. The problem of the QoS in the Internet requires a close cooperation between different protocols and components. For example, in the network there should be cooperation between the session setup protocols and the routing protocols to establish a path that can handle the QoS request. This also requires the cooperation of admission control, packet classifiers and packet schedulers.

In previous work [8] we have developed end-to-end resource reservations architecture. The architecture treats the Internet as a collection of autonomous systems that are configured and operated by the OSPF protocol. In OSPF, each autonomous system can be divided into areas interconnected by a backbone area. Therefore, we have adopted this configuration scheme and introduced a domain

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 67-78, 2000.

agency in each area (domain). In fact, if these areas are selected properly we can manage their network resources effectively, and obtain a better scalability for admitting immediate and advance resource reservations. In this paper we present a framework, which outlines the operation of the domain agencies in the OSPF configured networks.

Quality of service provision requires at least two processes; QoS negotiation process and resources setup process. The two processes require the exchange of messages between end systems (which may require the presence of the user) and between end systems and the network elements. The negotiation process continues in exchanging messages until a final decision is made. In fact, this process may take long time to finish, depending on the available resources.

An alternative way of achieving negotiation is to deploy the agent technology. Agents can work on behalf of the user or another agent to accomplish a specific task. In the negotiation process agents are provided with the necessary information (i.e. a set of policies) that enable them to act and accomplish their task without the users' involvement. We have considered two types of agents; static agents and mobile agents. Static agents accomplish their task, in the host node, without the need to move to another node, while mobile agents have the ability to move from one node to another in order to cooperate with other agents and/or software to accomplish their work. In fact, if those types of agents are used effectively in the network domains, the QoS negotiation and resources setup processes could be made much easier, scalable, flexible and fast.

The rest of this paper is organized as follows. In section 2, we give a brief description of the OSPF. Then we discuss the domain agency architecture in section 3. Section 4, presents the operation of the domain agencies in response to AS changes. We conclude the paper in section 5.

2 OSPF Domains

The open shortest path first (OSPF) [12] is an interior gateway protocol where it distributes routing information between routers belonging to a certain area. Each router maintains an identical database describing the area's topology. This database is referred to as the link-state database. The link-state database gives a complete description of the network: the routers and the network segments and how they are interconnected. OSPF allows collection of contiguous networks and hosts in an Autonomous System (AS) to be grouped together. Such a group, together with the routers having interfaces to any of the included networks is called an area. Each OSPF area is identified by a 32-bit Area ID and runs a separate copy of the basic link-state routing algorithm.

Routing in the AS takes place on two levels, depending on whether the source and destination of a packet reside in the same area (intra-area routing) or different areas (inter-area routing). It is interesting to note that since all routers have the same database, any router can calculate the routing table of any other router. This useful property has been taken into account in designing the QoS Routing Agent. Another interesting property is that the link-state database gives a complete description of the

network: the routers and network segments and how they are interconnected. We have also considered this property in designing the domain agency. Figure 1, presents an OSPF AS, which has been divided into areas.

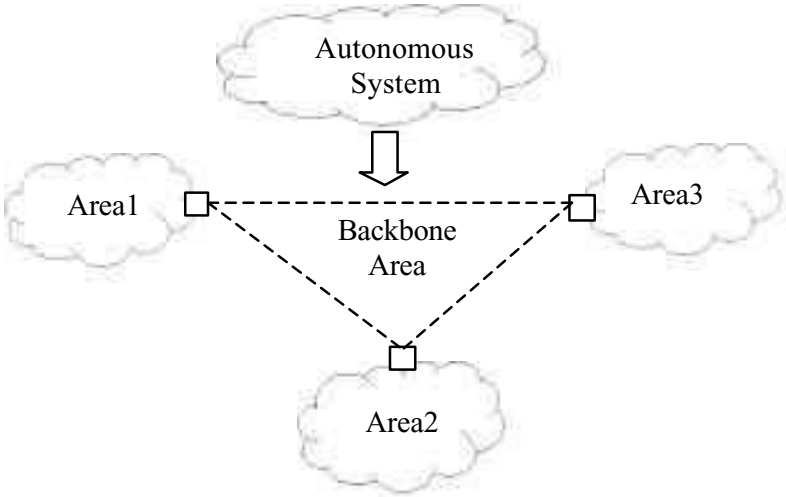
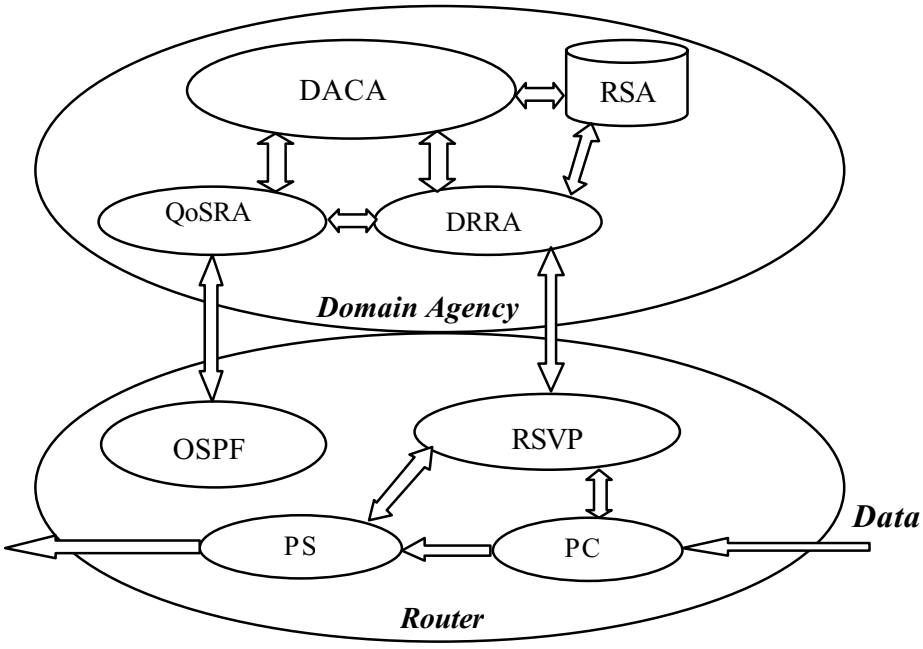


Fig. 1. OSPF areas

3 Domain Agency Architecture

The resource reservations architecture [8] relies on domain agencies, in network domains, to manage domain resources. The domain agency receives resource reservations requests from end systems and agencies in other domains. This architecture has been adopted for several reasons. The first reason is to provide better advance and immediate resource reservations scalability. In previous resource reservation architectures [15], every network element is responsible for reserving resources and maintaining them. Due to the high volume of resource reservations, this would overload the routers and affect their performance. The problem is getting even worse when advance reservations have been permitted for users, which encourage many users to reserve resources in advance. Therefore, by this architecture the agency will take care of resource reservations and maintain their states. The other reason for using this architecture is the QoS route calculations. Sessions that need specific QoS should send their requests to the domain agency, which calculates the best-path in the domain and reserve the necessary resources in this path. The traditional routing protocol works in its normal way for forwarding background traffic. This will relief the QoS routing protocol from finding the QoS routes on demand, as other proposals suggest [2,6,10,11,18,19]. Other reason for using domain agencies is that the domain agency takes care of any problems in the domain and hides them from the end systems. For example, route failure, the domain agency handles it without informing the end systems. The agency takes care of refresh messages as well, which allow us to control the flow of the refresh messages from end systems to the domain agencies and from the domain agencies to the routers within the domain. Figure 2 shows the

components of the domain agency. The agency should reside in any router except the area border routers and there should be a backup domain agency.



DACA: Domain admission control agent
DRRA: Domain resource reservations agent
OSPF: Open shortest path first
PC: Packet classifier
PS: Packet scheduler
QoSRA: Quality of service routing agent.
RSA: Reservations state agent
RSVP: Resource reservations protocol

Fig. 2. Domain agency architecture

The introduction of domain agencies in an AS is not an easy task. Areas in OSPF protocol are dynamic, where system administrators increase or decrease the number of areas in an AS as needed. Therefore, the domain agencies should adapt to this configuration changes and react quickly.

The reservation process as depicted in the resource reservation architecture [8] starts by sending a request from the receiver. Then the receiver and the sender engage in a negotiation process regarding the QoS and time of the playback. If this stage ends successfully, the next stage starts immediately, which is the resource reservation process. The flow of these messages is shown in Figure 3.

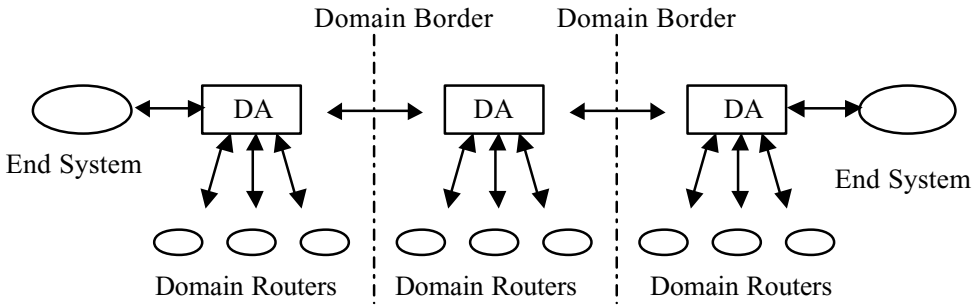


Fig. 3. flow of messages

The message types include the following:

Path Message: This message is sent from the sender to the domain agency in the sender domain. The message carries the traffic and time specifications, in addition to other information. After the agency finishes its job, it forwards the message to the next domain agency in the way to the receiver until it gets to the receiver.

Reserve Message: The receiver sends this message just after it receives the *Path Message* The message follows the reverse path of the *Path Message and* carries the same information. The *Reserve Message* is treated as a reserve confirmation message.

Start Message: Just before the playback of every session, the domain agency sends a start message to the routers involved in this session, to set some parameters in packet classifiers and packet schedulers.

Refresh Message: This message is sent from the receiver to maintain the reservations. The message is sent at certain time intervals to the domain agencies, and the agencies are responsible for sending refresh messages to the RSVP in the routers. Time-out of refresh messages will cause the domain agency to teardown the session.

User Interaction Message: This message is sent during the active or the passive state of the session. If there is any user interaction, the action is first negotiated between the receiver and the sender. If it is accepted, it is forwarded by the receiver to the domain agencies to update the reserved resources. The interaction request may come at a time where all resources are exploited, at this time the best-effort service can be used.

Session Teardown Message: This message can be initiated by the sender or by the receiver. It ends a specific session and releases the resources used by this session.

3.1 Domain Resource Reservation Agent

The Domain Resource Reservation Agent (DRRA) has interface to the RSVP, through which it receives and sends the messages. When the DRRA receives the *Path Message*, it submits traffic and time specifications to the QoS Routing Agent. The DRRA is also responsible for updating the reservations state agent. This update can be either adding accepted sessions or removing finished sessions. If the DRRA

receives any *User Interaction Message* it forwards it to the domain admission control agent.

3.2 Quality of Service Routing Agent

The Quality of Service Routing Agent (QoSRA) has been introduced to find the best-path that satisfies the QoS needs of the multimedia applications [7]. The QoSRA runs a modified dijkstra algorithm based on domain topology obtained from the OSPF, and the available link resources obtained from the domain admission control agent. The QoSRA is based on the OSPF property, which enables any router to calculate the routing table of any other router. Therefore, all resource reservations requests are sent to the domain agency that instructs the QoSRA to construct the QoS path. In fact, this is a new departure from other proposals [2,6,10,18,19] that suggest enhancing the existing routing protocols to support QoS. In these proposals the calculation is done by every router, which overloads the routers and make the advance reservations almost possible, because of scalability problem.

Two QoS routing calculation approaches are considered in the literature so far [11]. The first one is the on-demand calculation. In this approach the QoS path is calculated at the arrival of every request (requests that demand constraint QoS), this calculation is done by every router receiving the request. This approach is not scalable in the fast growing Internet and multimedia applications. This situation is even worse if the QoS requests include advance reservations. The second approach is to pre-compute paths to all destinations for each node. Then at the arrival of every request, the path that satisfies the requested QoS is selected to forward the session's packets. This approach has many disadvantages. The QoS paths should be frequently pre-computed to reflect the current available resources. For every arrived request, the QoS paths need to be searched for the suitable path, which adds extra processing. Due to the high demand on QoS, the pre-computed paths are out dated and don't reflect the currently existing resources.

3.3 Domain Admission Control Agent

The Domain Admission Control Agent (DACA) has two functions [7,9]. The first one is to find the available link resources of all links in the domain at specified time intervals and submit this information to the QoSRA. The second function is to receive user interaction requests from the DRRA and apply admission control algorithm.

The idea of monitoring the available link capacities is as follows. At the beginning the QoS routing agent receives the domain topology from the OSPF and submits it to the DRRA. The DRRA inquiries about each link capacity and submit the complete information to the reservations state agent. At each resource reservation request, the admission control finds the available link resources, for the specified time intervals, then give this information to the QoS routing agent. The QoS routing agent constructs the best-path for this request and submits the resulted path to the DRRA. The DRRA reserves the resources temporarily until it receives a confirmation message from the receiver.

3.4 Reservations State Agent

The Reservations State Agent (RSA) stores all accepted reservation states. For better performance, the RSA is divided into two parts. The first part keeps track of link reservation states, while the second part keeps track of aggregated link reservation states. Figure 4, shows a snapshot of resource reservation made in a single link. The numbers indicate session's id, while α indicates a new session asking for admission. As we can see in the figure the duration as well as the bandwidth of each session is clearly defined. Therefore, if we aggregate the reservation state of every link, we get a summary database as described in figure 5. This summary database enables the domain admission control agent to get quick values of the available link resources.

The RSA periodically checks the reservation database for due sessions, if there are any sessions ready to start, it sends the session's traffic specification to the RSVP in the routers. The RSVP then set some parameters in the packet classifier and packet scheduler to achieve the requested QoS. The time of the playback of the scenario is controlled by the RSA, where it sends a start and finish messages to the RSVP in the router.

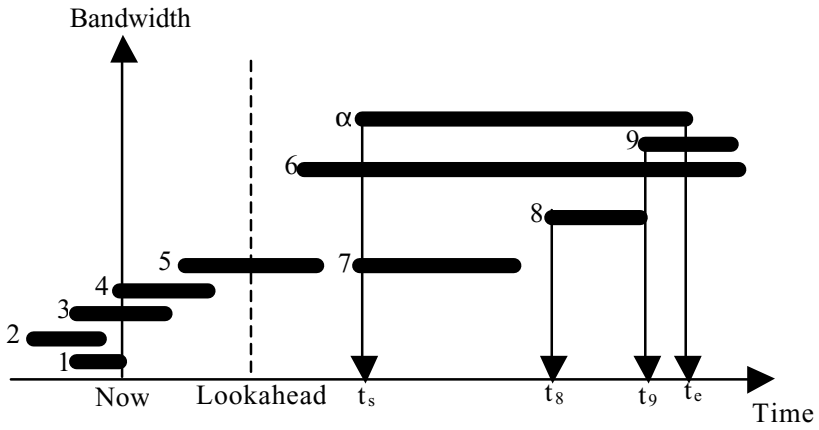


Fig. 4. snapshot of reservation state of a single link

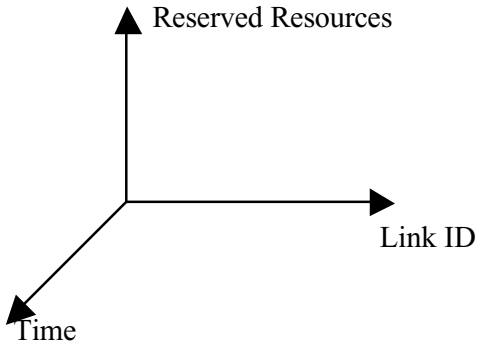


Fig. 5. dimensions of the reservation database

4 Domain Agency Communications Protocol

This section describes how domain agencies exchange messages and how agents in the same domain agency communicate with each other and with non-agent software. Then we state how domain agencies adapt to AS changes. For message exchange we use FIPA-OS as an agent platform. FIPA-OS (Foundation for Intelligent Physical Agents-Operating System) is a product from Nortel [20], which implements the FIPA specifications [21]. Figure 6, shows how the domain agency is mounted on top of the FIPA-OS. Agents in the domain agency communicate with the software (OSPF and RSVP) using agent wrappers. The OSPF agent wrapper gets information from the OSPF protocol and submits them to the QoSRA. The RSVP agent wrapper relays commands from the DRRA and RSA to the RSVP protocol, and forwards messages from RSVP to DDRA.

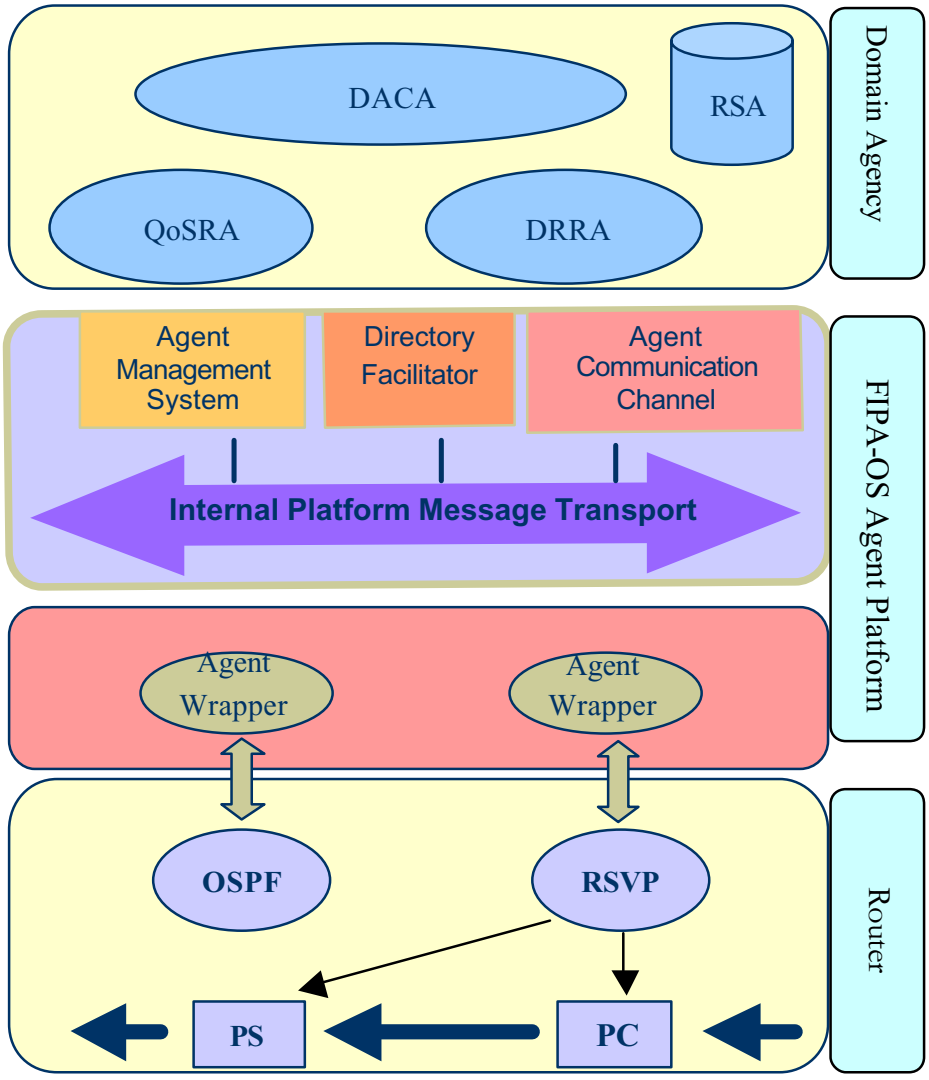


Fig. 6. Using FIPA-OS as an agent platform

4.1 Interactions between Domain Agencies

Assume that there is one AS and one DA that manages the whole AS resources. As soon as the AS is split into areas, the QoSRA will be aware of this separation from the OSPF. The QoSRA informs the DRRA with the new AS configuration. From this configuration, the DRRA knows the address and number of the new areas. Then it duplicates and sends one DA to each new area including the backbone (the backbone is an area with address 0.0.0.0). Each DA resides in a router other than the area border router.

After each DA settles in its domain, it needs to communicate back with the parent DA to get its information. The DRRA in the child DA gets the topology data from the QoSRA, and send this information to the parent DA. The parent DA gets the information related to each DA, according to topology data, and sends them back to their DA. At the same time the parent DA constructs addresses table for the new DAs, and send this table to every DA. This step is important to enable the DAs to communicate with each other.

The parent DA should fall into one of the new areas including the backbone. Figure 7, shows the parent DA in the backbone and other DAs communicate with it to fetch their information.

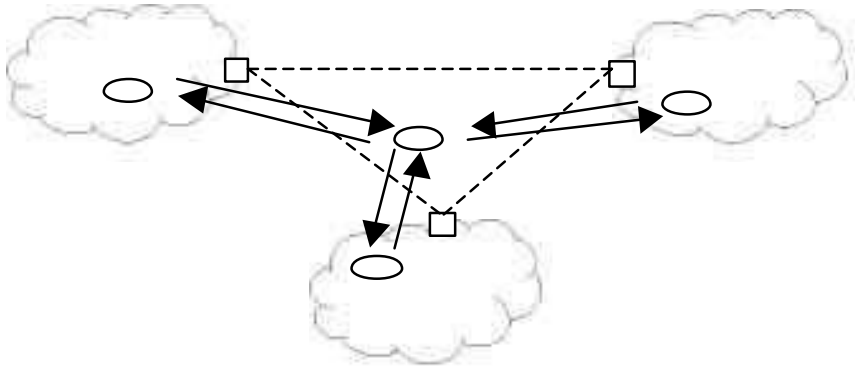


Fig. 7. Interactions between domain agencies and the parent DA

There are five packet types that are used to convey the information between domain agencies.

Database description packet: This packet is sent by the child DA to the parent DA. It carries the topology database of the child's domain.

Reservations State: For each child DA, the parent DA collects the reservations state from the reservations state agent and sends them back to the child DA. The message contains also the addresses table of the other DAs.

Policy and Service Agreement: Domain agencies need to cooperate with each other to setup policies that each DA should follow. DA could also request from another DA to reserve a certain path with certain bandwidth, this information is also communicated by this packet.

Hello Packet: Every DA should send Hello packets to its neighboring DAs, to maintain relationships and check reachability. The DAs should also send Hello packet to end systems, to identify themselves and maintain reachability.

Acknowledgment Packet: This packet is sent between the parent DA and the child DA. It has different interpretation as shown below:

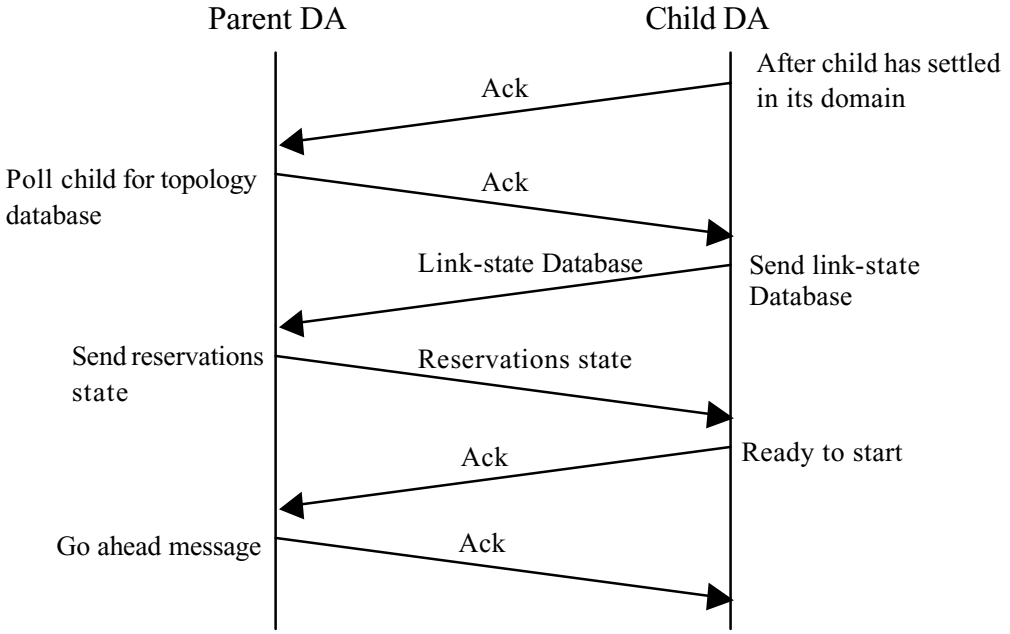
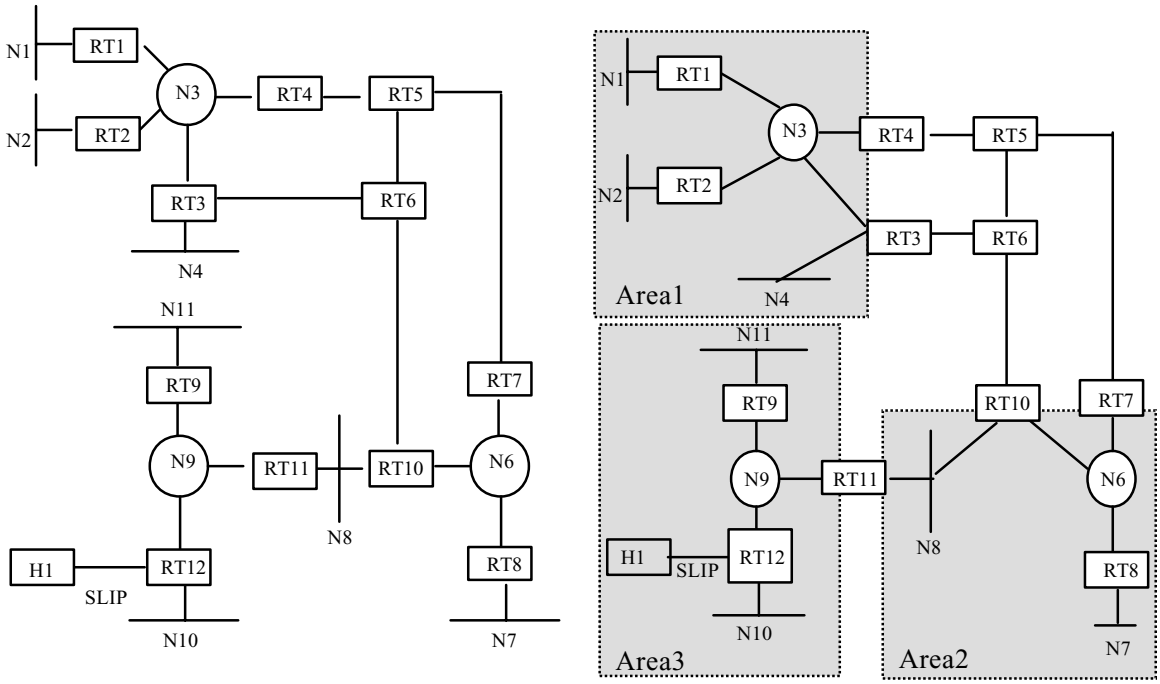


Fig. 8. Parent DA and child DA setup.

4.2 Status of Current (Immediate and Advance) Reservations

Figure 9, presents an AS before and after partition. In figure 9.a all network elements belong to the same area (AS). In this configuration, only one DA manages the AS resources. If this AS is split into several areas as shown in figure 9.b, the DA will react quickly and dispatch DA for each new area. All borders that have been introduced to create areas are virtual The location and interfaces of the network elements have not been changed. Therefore, the current resources reservations will also be applicable for the new configuration.



9.a. Before partition 9.b. After partition

Fig. 9. OSPF AS domain

5 Conclusion

Network domains provide a better chance for network protocol developers to scale their protocols and get control of the resources in the domain. The OSPF routing protocol has the advantage of dividing an autonomous system into manageable areas, where each node in the domain knows only the nodes in its domain. We have considered this advantage in designing our resource reservation architecture and assigned a domain agency in every domain. In this paper, we have provided a framework, which enable domain agencies to react to area splitting and communicate with each other and with the end systems.

References

1. A. Hafid, G. V. Bockmann and R. Dssouli, “A Quality of Service Negotiation Approach with Future Reservation (NAFUR): A Detailed Study,” Technical Report, University of Montreal, Montreal, 1996.

2. A. Shaikh, J. Rexford, and K. Shin, "Efficient Precomputation of Quality-of-Service Routes," Proc. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '98), July 1998, Cambridge, England, pp. 15-27.

3. Aurrecoechea, C., Campbell, A.T. and L. Hauw, "A Survey of QoS Architectures", ACM/Springer Verlag Multimedia Systems Journal , Special Issue on QoS Architecture, Vol. 6 No. 3, pg. 138-151, May 1998.
4. D. D. Clark, S. J. Shenker, and L. Zhang, “Support Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism,” Proc. Of ACM SIGCOMM’92, Aug. 1992, pp. 14-26.
5. D. Ferrari, A. Gupta and G. Vertre, “Distributed Advance Reservation of Real-Time connections,” Fifth International Workshop on Network and operating system Support for Digital Audio and Video, Durham, NH, USA, April 19-21,1995
6. E. Crawley, R. Nair, B. Rajagopolan, H. Sandick, "A Framework for QoS-Based Routing in the Internet" RFC 2386, August 1998.
7. F. Sallabi, A. Karmouch, "Immediate and Advance Resource Reservations Architecture with Quality of Service Routing Agents" Proceedings of Multimedia Modeling (MMM’99), October 4-6, 1999, Ottawa, Canada.
8. F. Sallabi, A. Karmouch, "New Resource Reservation Architecture with User Interactions" Proceedings of IEEE Pacific Rim Conference on Communications, Computer and Signal Processing, August 22-24, 1999, Victoria, B.C, Canada.
9. F. Sallabi, A. Karmouch, " Resource Reservation Admission Control Algorithm with User Interactions " Proceedings of the Globecom’99, December 5-9, 1999, Rio de Janeiro, Brazil.
10. G. Apostolopoulos, R. Guerin, S. Kamat, A. Orda, and S. Tripathi. ``Intra-Domain QoS Routing in IP Networks: A Feasibility and Cost/Benefit Analysis." Special Issue of IEEE Networks on Integrated and Differentiated Services for the Internet, September 1999.
11. G. Apostolopoulos, R. Guerin, S. Kamat, A.Orda, T. Przygienda, and D. Williams. "QoS Routing Mechanisms and OSPF Extensions." RFC 2676, Experimental RFC, Internet Engineering Task Force, August 1999.
12. J. Moy, "OSFP Version 2" RFC 2328, April 1998.
13. L. Wolf, L. Delgrossi, R. Steinmetz, S. Schaller, and H. Witting, “Issues of Reserving Resources in Advance,” In Proceedings of NOSSDAV, Lecture Notes in Computer Science, Pages 27-37, Durham, New Hampshire, April 1995, Springer.
14. O. Schelen and S. Pink, “Sharing Resources through Advance Reservation Agents. In Proceedings of IFIP Fifth International Workshop on Quality of Service (IWQoS’97), New York, May 1997, pp. 265-276.
15. R. Braden, L. Zhang, S. Berson, S. Herzog and S. Jamin, “Resource Reservation Protocol (RSVP)—Version1 Functional specification,” ftp://ftp.ietf.org/internet-drafts-ietf-rsvp-spec-16.ps.
16. S. Berson, R. Lindell, and R. Braden, “An Architecture for Advance Reservations in the Internet,” Technical Report, USC Information Sciences Institute, July 16, 1998.
17. W. Reinhardt, “Advance Reservation of Network Resources for Multimedia Applications,” Proceedings of the Second International Workshop on Advanced Teleservices and High Speed Communication Architectures, 26-28 September 1994, Heidelberg, Germany.
18. Z. Whang and J. Crowcroft, "Quality of Service Routing for Supporting Multimedia Applications" IEEE Journal on Selected Areas in Communications, 14(7): 1228-1234, September 1996.
19. Z. Zhang, C. Sonchez, B. Salkewicz, and E. Crawley, "Quality of Service Extensions to OSPF or Quality of Service Path first Routing (QOSPF)" Internet draft (draft-zhang-qos-ospf-01.txt", work in progress, September 1997.
20. http://www.nortelnetworks.com/products/announcements/fipa/
21. http://www.fipa.org/

Partitioning Applications with Agents

Oskari Koskimies and Kimmo Raatikainen

Department of Computer Science
P.O. Box 26 (Teollisuusk atu 23)
FIN-00014 UNIVERSITY OF HELSINKI
Firstname.Lastname@cs.helsinki.fi
http://www.cs.helsinki.fi/research/monads/

Abstract. The environment of mobile computing is in many respects very different from the environment of the traditional distributed systems of today. Bandwidth, latency and delay may change dramatically when a nomadic end-user moves from one location to another or from one computing environment to another. The variety of terminal devices which nomadic users use to access Internet services also increases at a growing rate.
Dynamic adaptation of a service to the properties of terminal equipment and a vailable communication infrastructure is an attractive feature. With application partitioning, an application consisting of co-operating component agents can be dynamically distributed on both sides of the wireless link. By selecting a partitioning configuration based on terminal characteristics, an application can be adapted to the capabilities of the terminal. Partitioning can also be used for adapting to wireless link quality, by repartitioning the application when link quality changes sufficiently. We have designed a service for performing the partitioning decisions, and used a prototype implementation to prove that the communication delays incurred by repartitioning are acceptable.

1 Introduction

The environment of mobile computing is in many respects very different from the environment of the traditional distributed systems of today. Bandwidth, latency, delay, error rate, interference, interoperability, computing power, quality of display, and other non-functional parameters may change dramatically when a nomadic end-user moves from one location to another, or from one computing environment to another – for example from a wired LAN via a wireless LAN[3] (WLAN) to a GPRS[11] or UMTS[17] network. The variety of mobile workstations, handheld devices, and smart phones, which nomadic users use to access Internet services, increases at a growing rate. The CPU power, the quality of display, the amount of memory, software (e.g. operating system, applications), hardware configuration (e.g. printers, CDs), among other things ranges from a very low performance equipment (e.g. hand held organizer, PDA) up to very high performance laptop PCs. All these cause new demands for adaptability of data services. For example, palmtop PCs cannot properly display high quality

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 79-93, 2000.

images designed to be looked at on high resolution displays, and as nomadic users will be charged based on the amount of data transmitted over the GPRS network, they will have to pay for bits that are totally useless for them.

Software agent technology has gained a lot of interest in the recent years. It is widely regarded as a promising tool that may solve many current problems met in mobile distributed systems. However, agent technology has not yet been extensively studied in the context of nomadic users, which exhibits a unique problem space. The nomadic end-user would benefit from having the following functionality provided by the infrastructure: Information about expected performance provided by agents, intelligent agents controlling the transfer operations, a condition-based control policy, capability provided by intelligent agents to work in a disconnected mode, advanced error recovery methods, and adaptability.

The research project Monads [10] examines adaptation agents for nomadic users [15]. In the project we have designed a software architecture based on agents and we are currently implementing its prototypes. Our goal is not to develop a new agent system; instead, we are extending existing systems with mobility-oriented features. The Monads architecture is based on the Mowgli communications architecture [14] that takes care of data transmission issues in wireless environments. In addition, we have made use of existing solutions, such as FIPA specifications [5] and Java RMI [16], as far as possible. However, direct use was not sufficient but enhancements for wireless environments were necessary [2, 13].

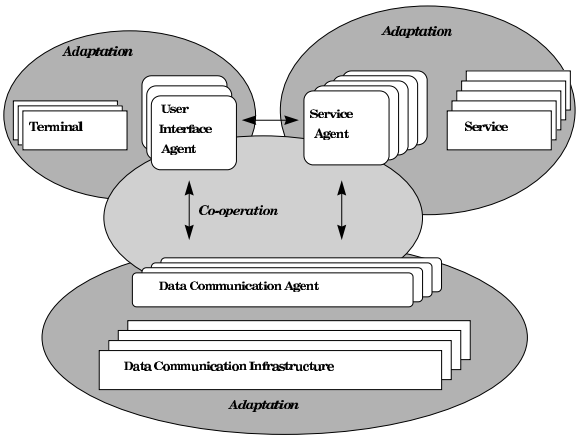


Fig. 1. The Adaptation Triad

By adaptability we primarily mean the ways in which services adapt themselves to properties of terminal equipment and to characteristics of communications. This involves both mobile and intelligent agents as well as learning and predicting temporary changes in the available Quality-of-Service (QoS) along the communications paths. The fundamental challenge in nomadic computing is dynamic adaptation in the triad service–terminal–connectivity (see Figure 1) according to preferences of the end-user.

The ability to automatically adjust to changes in the wireless environment in a transparent and integrated fashion is essential for nomadicity – nomadic end-users are usually professionals in other areas than computing. Furthermore, today's distributed systems are already very complex to use as a productive tool; thus, nomadic end-users need all the support, which an agent based distributed system could deliver. Adaptability to the changes in the environment of nomadic end-users is the key issue. Intelligent agents could play a significant role in implementing adaptability. One agent alone is not always able to make the decision how to adapt, and therefore adaptation is a co-operation effort carried out by several agents. Thus, there should be at least some level of cooperation between adapting agents.

Dynamic adaptation of a service to the properties of terminal equipment and available communication infrastructure is an attractive feature. We have previously explored predictive adaptation to available bandwidth with a Web browsing agent [15]: When the network connection is slow or predicted to become slow, the browser agent may automatically use different kinds of compression methods or even refuse to fetch certain objects.

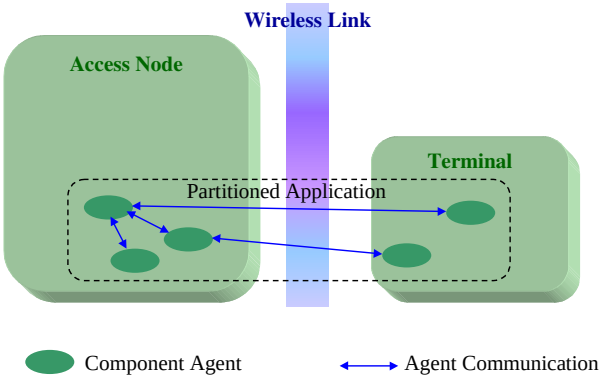


Fig. 2. Application partitioning

We are now also examining adaptation to terminal equipment through *application partitioning*. With application partitioning we refer to the idea of dividing an application into component agents that communicate using e.g. FIPA ACL[7] messages. In this way, the application is running in a distributed fashion on both sides of the wireless link, as depicted in Figure 2.

Partitioning can be either *static*, *partially dynamic* or *fully dynamic*. *Static* partitioning, where component agent configuration is determined at compile time and cannot be changed, is of little interest to us. In *partially dynamic* partitioning, the location of a component agent is determined dynamically during application initialization, but cannot change during the application session. The most interesting is *fully dynamic* partitioning, which allows the component agents to be moved at any time during the application session. This is useful when bandwidth (or other dynamic resources such as memory) changes radically, and the

change is expected to last for some time. For bandwidth, such a drastic change would typically be a vertical handover or a disconnection.

Partially dynamic partitioning is sufficient for adapting to different terminal types, but adapting to bandwidth changes (including disconnections) requires fully dynamic partitioning. Note that fully dynamic partitioning requires the component agents to be mobile, whereas partially dynamic partitioning does not.

2 The Need for Terminal Adaptation

The QoS of a wireless link can vary wildly, due to interference, network load and vertical handovers. In some cases, like vertical handover, these changes can be quite drastic. In order to provide smooth operation, an application needs to adapt to changes in QoS. Traditionally, the adaptation is done by compressing and reducing content that is transferred to the terminal.

However, adapting to just QoS is not enough. Unlike the users of fixed networks, who almost all have fairly similar, workstation-class terminals, the terminals of nomadic users vary from smartphones to powerful laptops. Even the same user is likely to use different classes of terminals, e.g. a laptop on business trips and a PDA or smartphone during his free time. However, currently the nomadic user often has to use a different application for the same purpose on different classes of terminals. For example, the calendar software on a laptop is likely to be very sophisticated and offer an advanced graphical user interface, whereas the software on a smartphone would probably offer only minimal functionality. Even though the different applications may be able to exchange information, the situation is still undesirable for two reasons:

1. An application is limited to the terminal class it was designed for, and
2. Users have to learn a different application for each terminal class.

The application can, of course, be implemented separately for each terminal class, but this wastes implementation effort, and a version for smaller terminals may have drastically reduced functionality. What is required is an application that can adapt to different terminal classes without sacrificing functionality[1].

3 Adaptation by Partitioning

We are examining adaptation to terminal equipment and QoS variation by partitioning an application into components. Because adaptation of the overall application requires that the individual components can cope with varying component configurations, the components must both be themselves adaptive, and also have a degree of autonomy. Thus, it makes obvious sense to model the components as

[1] However, *usability* may suffer. No amount of adaptation can make the keyboard or screen of a smartphone match that of a laptop.

agents. To give an example: An email application can work with different types of terminals by partitioning the application in different ways, depending on terminal and wireless link characteristics. Assume the email application consists of four agents:

1. *User interface agent*: Either a standard FIPA UDMA agent [8] or a dedicated UI agent that has an advanced graphical user interface and supports voice input.
2. *Core email agent*: Handles basic email processing.
3. *Email filtering agent*: Groups and prioritizes messages, and handles any automated email processing.
4. *Email compression agent*: Compresses messages prior to transferring them to the terminal (this includes lossy methods such as leaving out attachments).

With low-performance equipment (a high-end mobile phone, for example), only the user interface (a standard FIPA UDMA agent) of the application runs on the terminal. If the terminal has more capabilities (like a PDA), also the core email agent can run on the terminal, with the email filtering and compression agents running on the network side. Finally, with a laptop terminal, all agents except the compression agent can be run on the terminal.

Partitioning can be used for adapting to available bandwidth as well. For example, if bandwidth is very low, it is better to run the filtering agent on the network side, since it may be able to handle some emails automatically, and prioritizing means the user will get the more important messages first. On the other hand, if bandwidth is high, it is better to run the filtering agent on the terminal, since that allows it to more easily communicate with the user about filtering decisions, enabling more fine-grained filtering control. By using mobile agents, this kind of adaptation can be dynamic, with agents moving to optimal locations while the application is running. We call this *repartitioning*.

Another facet of partitioning is the possibility to use different agents for different terminals, or not to use a particular agent at all. For example, with a high-performance laptop, a large, dedicated email UI agent can be used instead of a small, generic FIPA UDMA agent. Or, when bandwidth is high, the use of an email compression agent becomes unnecessary.

3.1 Assumptions

The design of our partitioning system is based on a few assumptions:

Applications are specially designed We assume that the application is specially designed to support partitioning, for the following reasons:

1. An application that wasn't designed to be partitioned is unlikely to be easily divided into separate independent components.
2. Partitioning, and especially repartitioning, requires the component agents to be designed flexible so that reconfiguration is possible. It is unrealistic to expect this from applications in general, although applications built by component composition (from "off-the-shelf" component agents) in the future might have the required flexibility.

3. Practical agent applications are still scarce, so one can assume that the most important agent applications are still to be built, and their design is thus still open.

Application metadata is available Since applications are already assumed to be specially designed for partitioning, it is not unreasonable to assume that some metadata (such as agent resource requirements) has been made available as well.

Repartitioning is heavy and uncommon The actual time taken by any single repartitioning operation depends on available bandwidth and the size of the agents involved, but we assume that repartitioning is a heavy operation, due to both its transactional nature and the need to transfer agent state. Thus, repartitioning would be worthwhile only when drastic changes in circumstances occur (e.g. vertical handover).

Note that the initialization of a partitioned application is a much lighter operation, but may still require application code to be transferred over the wireless link.

3.2 How to Partition an Application

Our project has previously produced a system for predicting near-future QoS fluctuations[15]. We utilize these predictions, along with profiles, to make partitioning decisions.

Three types of profiles are used:

Application profiles The Application profile lists the component agents in the application, and a set of possible configurations for them. For each configuration, agent locations and a communication profile are given, as well as an *utility* value that represents how "good" (in terms of usability) that configuration is for the user. If the application is willing to share some agents with other applications, that information would be here also.

Note that alternative (such as different GUI agents) and optional (such as compression agents) agents can be represented by configurations where all agents are not included.

Agent profiles Agent profiles contain resource requirements for the agent, as well as startup cost and repartitioning (movement) costs. Additionally, there are three flags:

- *External Agent*: The agent does not understand partitioning messages. It can still be used in a partitioned application, but the other agents have the responsibility for ensuring that its state is preserved across repartitioning. From the point of view of the partitioning service this flag simply means that the agent will be excluded from partitioning processes except for telling it to move to a new location or to shut down.
- *Movable*: The agent is mobile. If it is an external agent, its movement must be accomplished through normal agent management operations.
- *Replaceable*: The agent can be replaced by another agent. This requires that the agent is either stateless or saves its state to other agents before repartitioning.

Note that an agent might be neither movable nor replaceable. For these agents, relocation is either not possible or not desirable. Repartitionings that would affect these agents are not possible.

Terminal profiles The capabilities of terminals, such as memory and screen size, are listed in terminal profiles.

Figure 3 illustrates how a partitioning decision is made. When an application is started, the partitioning service first loads the application profile. The list of agents is then used to get the resource requirement profiles for the agents. These requirements are compared against the terminal capability profile to get a list of possible configurations for this terminal, and the communication profiles for those configurations.

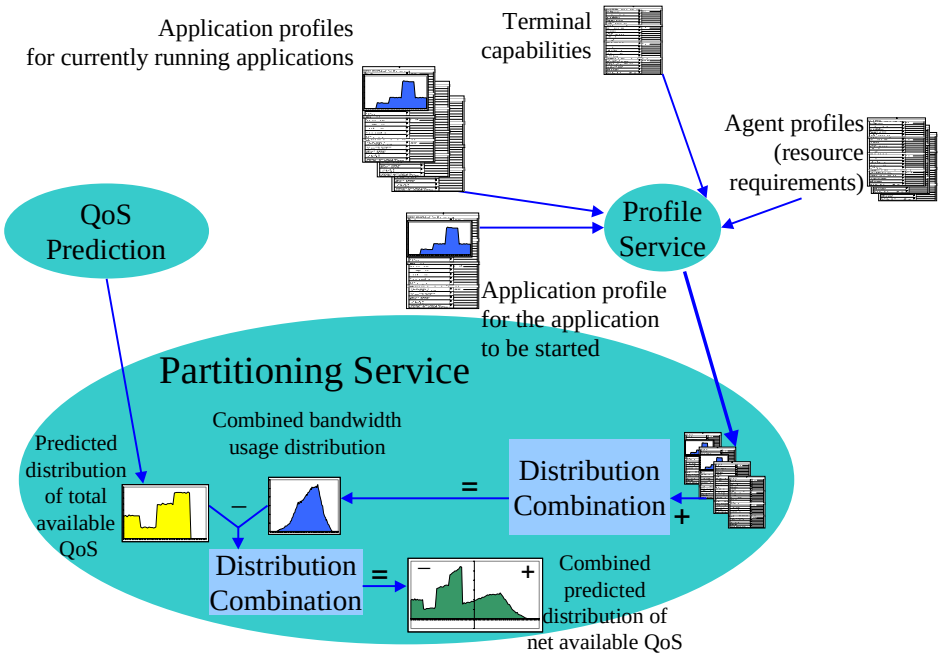


Fig. 3. Partitioning decision

In simple terms, for each possible application configuration, its bandwidth use (from the communication profile) is added to the bandwidth use of currently running applications, to get combined bandwidth use. This is then deducted from the predicted total available bandwidth, to get the predicted net available bandwidth. This is the prediction of the remaining bandwidth after the application is started.

The reality is only slightly more complex: A prediction of available bandwidth is not a single value but a distribution that gives the probability for getting a specific bandwidth. Likewise, the bandwidth use of an application varies according to a probability distribution. Thus, the result of the above calculation is also a distribution.

If the resulting distribution gives a high probability for having a negative net (remaining) bandwidth, that means that the configuration that is being examined cannot be run with the available bandwidth. Otherwise, the resulting distribution is given an *utility score* that is based on how much bandwidth was taken by it, and the utility value of that configuration for the user (available from the application profile). The score is penalized if the configuration exhausts available resources on the terminal. The configuration with the best final score is then selected.

3.3 Starting the Application

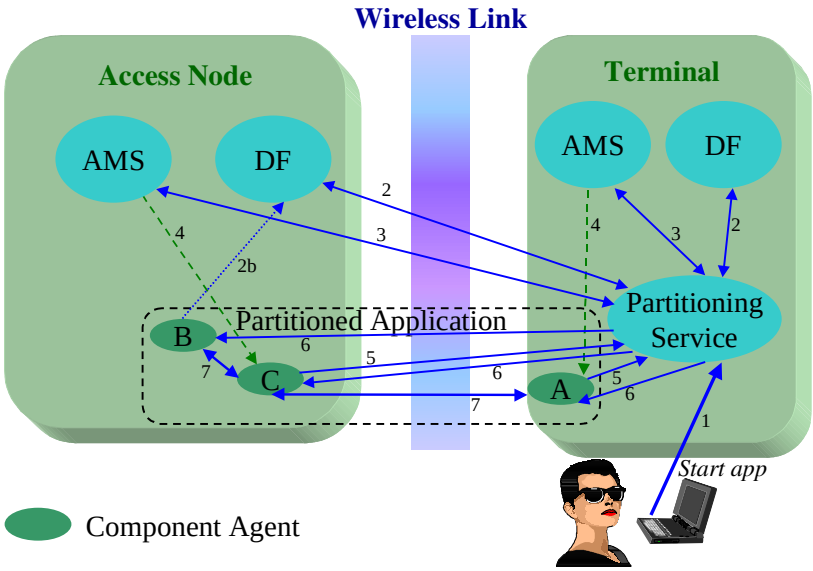


Fig. 4. Application startup

The application startup sequence is shown in Figure 4. The original startup request (1) is directed to the partitioning service. First, the partitioning service selects a unique ID for the application session. Then it queries the yellow pages (e.g. a FIPA DF[6]) in the terminal and on the network side (2) to see if some agents are already running and previously registered (2b), and then issues create-agent requests for those agents that are not running (or cannot be shared) (3).

To start an agent, the partitioning service may use a Factory service that advertises itself via the yellow pages, or it can contact the agent platform directly (e.g. a FIPA AMS[6]). Either way, a new agent is created (4). If agent profiles contain a startup cost, the fact that an agent is running can also be taken into account when selecting configurations. When an agent has successfully initialized itself, it sends a message (5) to the partitioning service to confirm that it is ready.

Finally, the application session ID is sent to all participating agents, together with information about their partners (6), binding them together to form the

application. The application session ID is especially important for shared agents, who use it to manage their state information. Application communication can now start (7). In fully dynamic partitioning, this phase can be redone during repartitioning to remove the need for rerouting messages of moved agents.

3.4 Repartitioning

The weakness of partially dynamic partitioning is that while the partitioning decision depends on both terminal capabilities and the QoS of the wireless connection, only terminal capabilities are relatively static. QoS may change drastically during the lifetime of an application session, making a previously made partitioning decision invalid. Fully dynamic partitioning becomes necessary: The application must be *repartitioned* by moving its component agents to new locations. Thus, mobile agents are required.

In fully dynamic partitioning, the decisionmaking process is rerun when QoS is predicted to change drastically. Frequent reruns of the process can be avoided by ignoring smaller changes, or by only considering vertical handoffs and disconnections.

If a rerun of the decisionmaking process shows that another configuration is superior to the current one, the difference of the utility scores of the configurations (the profit), multiplied with the time the new conditions are predicted to last, is compared to a penalty calculated from the repartitioning costs of the agents. If the profit is greater, repartitioning is initialized.

The actual repartitioning process, shown in Figure 5, is somewhat like a two-phase commit (see e.g. [9]). First, the partitioning service sends each component agent a partitioning request that contains the application session ID and the agent's orders (to stay, move, be replaced or shut down), and asks if the agent is able to participate. The agent checks if it can free itself of any application state that cannot be carried over the repartitioning process, and sends a `yes` answer if it can. Once the answer has been sent, the agent enters a state where it waits for the partitioning process to complete, refusing requests not related to the partitioning process. Note that even though an agent answers `no`, it should not continue normal operation, since other agents in the application may not be ready to continue yet.

Error handling follows the normal two-phase commit procedure, with one exception: An agent may be unable to move because it is shared by other applications. In that case, it sends back a `no` answer, and additionally indicates that the refusal is due to a sharing violation. The partitioning service may then immediately (without aborting) resend the partitioning request to the agent, but with replacement orders instead of movement orders. Since the additional message exchange takes time, service-type shared agents should preferrably be marked as non-movable in the agent profile.

If the partitioning service gets a `yes` answer from all agents, it then sends each agent a message which tells them what to do. Once an agent has completed the order, it sends back a reply to the partitioning service to signal that the order has been successfully executed. If the agent platform does not support

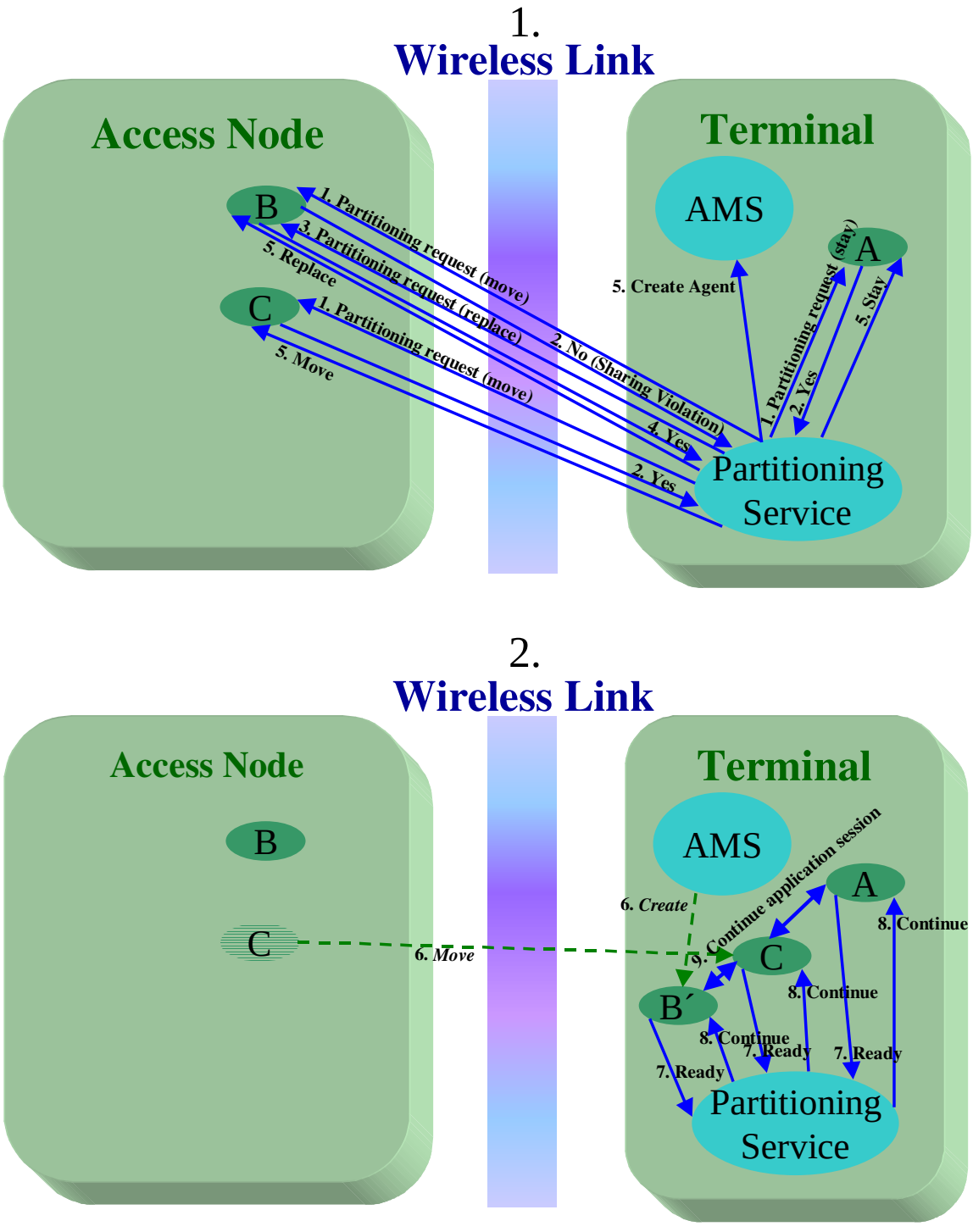


Fig. 5. Repartitioning

adequate message rerouting after agent movement, the reply may contain the new messaging address of the agent.

If the new configuration requires agents that were not present in the old one, these are created the same way as in application startup, described in the previous section. If an agent in the old configuration is not present in the new one, its order will be to shut down, and it will send the reply just before shutting down.

Finally, once all agents have reported in, the partitioning service sends each agent a `continue` message to tell them that the repartitioning has completed, and informs them of changed messaging addresses and new, removed or replaced agents. The agents can now continue from where they left off. Note that in some cases this is not straightforward; for example, a simple user interface agent may have been replaced with a more complex one, or some agents may not be present in the new configuration. However, it is up to the agents themselves to adapt to the new situation.

3.5 Repartitioning and Personal Mobility

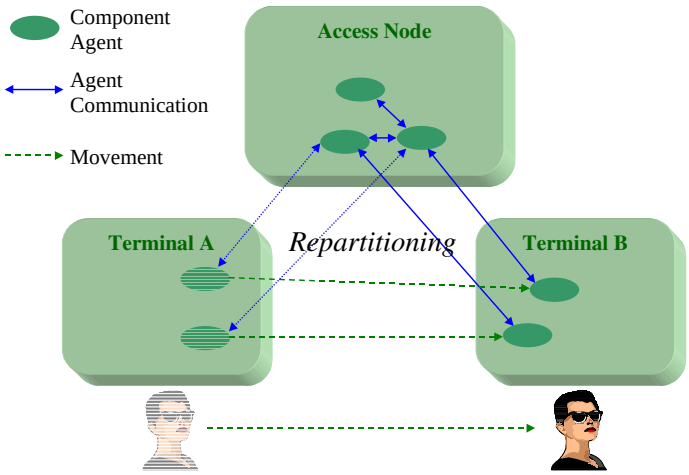


Fig. 6. Personal Mobility

As shown by Figure 6, repartitioning can also be used for personal mobility. This requires only minor extensions to the system described above. Firstly, the partitioning service on terminal A must fetch the terminal profile for terminal B, and use it instead of the terminal A profile. Secondly, all agents that the repartitioning process would have placed on terminal A, are issued orders to move to terminal B. Agents that are specific to a terminal type may be replaced by others, but that is a normal part of the partitioning process. Finally, the partitioning service on terminal A must pass on the responsibility for the application to the partitioning service on terminal B.

3.6 An Example Scenario

To give an idea of the communication costs involved in partitioning, we have built a prototype of the partitioning system, using the JADE 1.4 [1, 4] agent platform[2]. It was tested with a test application consisting of four agents, mimicing the email application example. Each agent had only the functionality necessary for the partitioning process, and some dummy state data.

Table 1. Example Agents

Agent	Description	State Information	Footprint
UDMA	Generic user interface agent	10 kB	100 kB [3]
DGUI	Dedicated UI	10 kB	1 MB
Core	Core Email Agent	100 kB	1 MB
Filter	Email Filtering Agent	20 kB	200 kB
Compr	Email Compression Agent	–	200 kB

The agents, their state data and memory requirements (as contained by the profiles) are described in Table 1. The test scenario is outlined in Figure 7:

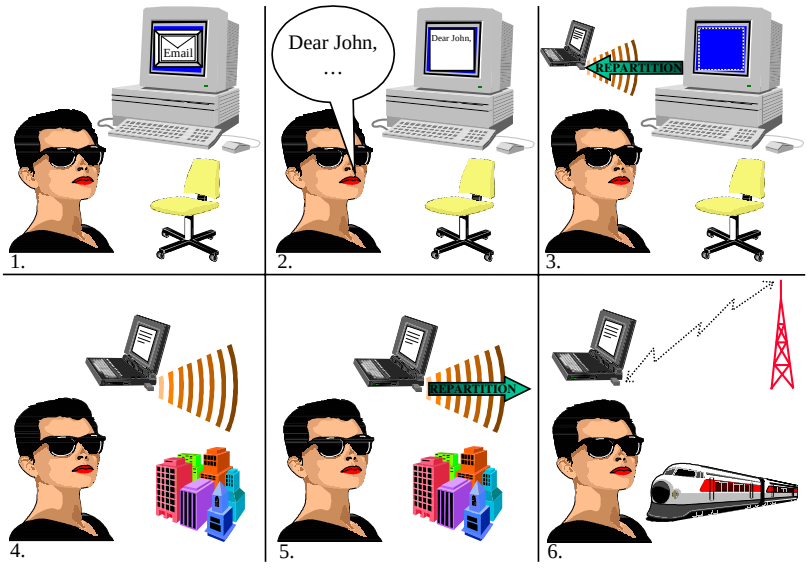


Fig. 7. Example Scenario

[2] The reason for using Jade is that it is also otherwise used by our project. However, JADE's implementation of agent mobility is not optimized, so using it also has the benefit of getting conservative results.

[3] The generic user interface agent can be shared by other applications.

1. The user is in her office, using a desktop computer with a 100 Mbps LAN connection. She starts reading her mail. The partitioning system chooses a configuration with a dedicated UI agent and without an email compression agent, where everything is running on the terminal, and starts the email application.
2. The user reads her mail. After a while, she comes across one that requires a reply, and switches to voice input, dictating the reply message.
3. After dictating about half of the reply, she notices it is time to go home, and switches to her palmtop, which has a 2 Mbps WLAN connection in the office, and a 28.8 kbps GSM High Speed Data connection[12] while outside[4]. The partitioning system moves the email application to the palmtop, using now a configuration with a generic user interface agent. The partial reply is saved to the state of the core email agent, and moved with it.
4. The user finishes writing the reply while waiting for the elevator. Because a generic UI agent is used, the GUI of the email application is now less polished, and no longer accepts voice input, but the user can continue to write the reply using normal text input.
5. Based on the time, and the fact that the user left the office, the QoS prediction system[15] gives a high probability that bandwidth will soon drop dramatically. The partitioning system recalculates the optimal configuration. The resulting configuration has an email compression agent and the email filter agent on the network side, and the other agents in the terminal.
6. The user continues to read her email using this configuration while sitting in a train. Although the user interface of the email application is more bare (generic UI), attached images lose some detail or are omitted (compression), and messages from mailing lists are ignored (filtering), she is still using the same application, and even the same application session, as when she started reading her email.

The scenario was tested with our prototype. The test was run using a Pentium III Linux workstation as the access node, a Pentium II Linux workstation as the desktop terminal and a Pentium Linux laptop as the palmtop computer. A real WLAN was used, but the GSM data link was simulated with software. The results are given in Table 2. Note that the times are median values from five test runs.

Table 2. Test Results

Event	Bandwidth	State data	Time
Application start (1)	100 Mbps	–	0.7 s
Terminal change repartitioning (3)	2 Mbps	120 kB	4.3 s
QoS change repartitioning (5)	2 Mbps	20 kB	2.2 s

[4] All these connection types are already commercially available.

As can be seen, the communication delays are quite acceptable for this scenario. On the last partitioning, however, that is due to the successful QoS prediction that allowed repartitioning to be initiated while bandwidth was still high. If the repartitioning had been done *after* the drop in QoS, when bandwidth was down to 28.8 kbps, the repartitioning would have taken 14 seconds – still acceptable, but a very noticeable delay. Note that application classes were already present on access node and terminals.

4 Conclusions and Future Work

We have shown how partitioning can be used to adapt an application to varying QoS and terminal capabilities. Partitioning can be used for implementing personal mobility, as well. Our first tests indicate that the solution is feasible as far as communication costs are concerned.

Our next step will be to fully implement the partitioning system and use it in the Monads QoS prediction system, so that the prediction system can be optimally configured for different terminal types. The problem of how to minimize state loss during repartitioning is also worthy of attention. Finally, partitioning-related messaging must be optimized to reduce the partitioning overhead.

References

1. F. Bellifemine, G. Rimassa, and A. Poggi. JADE – A FIPA-compliant Agent Framework. In *Proceedings of the Fourth International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents (PAAM 1999)*, April 1999.
2. S. Campadello, H. Helin, O. Koskimies, and K. Raatikainen. Performance Enhancing Proxies for Java2 RMI over Slow Wireless Links. In *Proceedings of the Second International Conference and Exhibition on the Practical Applicaiton of Java (PA Java 2000)*, April 2000.
3. B. P. Crow, I. Widjaja, J. G. Kim, and P. T. Sakai. IEEE 802.11 Wireless Local Area Networks. *IEEE Comm. Mag.*, pages 116–126, September 1997.
4. Jade Web Site. Available electronically from `http://sharon.cselt.it/projects/jade/`.
5. Foundation for Intelligent Physical Agents. FIPA Web Site. Available electronically from `http://www.fipa.org/`.
6. Foundation for Intelligent Physical Agents. FIPA 97 Specification Part 1: Agent Management, October 1997. Available electronically from `http://www.fipa.org/`.
7. Foundation for Intelligent Physical Agents. FIPA 97 Specification Part 2: Agent Communication Language, November 1997. Available electronically from `http://www.fipa.org/`.
8. Foundation for Intelligent Physical Agents. FIPA 98 Specification: Human-Agent Interaction, 1998. Available electronically from `http://www.fipa.org/`.
9. A. Goscinski. *Distributed Operating Systems: The Logical Design*, chapter 5.4.8, pages 203–204. Addison-Wesley, 1991.
10. Monads Research Group. Monads Web Site. Available electronically from `http://www.cs.helsinki.fi/research/monads/`.

11. GSM Technical Specification, GSM 02.60. GPRS Service Description, Stage 1, 1998. Version 6.1.0.
12. GSM Technical Specification, GSM 03.34. High Speed Circuit Swithed Data (HSCSD), Stage 2, May 1999. Version 5.2.0.
13. H. Helin, H. Laamanen, and K. Raatikainen. Mobile Agent Communication in Wireless Networks. In *Proceedings of the European Wireless'99 Conference*, October 1999.
14. M. Kojo, K. Raatikainen, M.Liljeberg, J.Kiiskinen, and T.Alanko. An Efficient Transport Service for Slow Wireless Telephone Links. *IEEE Journal on Selected Areas in Communications*, 15(7):1337–1348, September 1997.
15. M. Mäkelä, O. Koskimies, P. Misikangas, and K. Raatikainen. Adaptability for Seamless Roaming Using Software Agents. In *XIII International Symposium on Services and Local Access (ISSLS2000)*, Stockholm, Sweden, June 2000.
16. Sun Microsystems. Java Remote Method Invocation – Distributed Computing for Java. White Paper, 1998.
17. Third Generation Partnership Project Web Site. Available electronically from `http://www.3gpp.org/`.

Evaluating the Network Performance Management Based on Mobile Agents

Marcelo G. Rubinstein[1,2,⋆], Otto Carlos M. B. Duarte[1], and Guy Pujolle[2]

[1] Grupo de Teleinformática e Automação, Universidade Federal do Rio de Janeiro, COPPE/EE, CP 68504, 21945-970, Rio de Janeiro RJ, Brazil,
{rubi, otto}@gta.ufrj.br
[2] Laboratoire PRiSM, Université de Versailles,
45, Avenue des Etats-Unis, 78035, Versailles Cedex, France
Guy.Pujolle@prism.uvsq.fr

Abstract This paper aims to investigate mobile agent scalability in network management, in order to find when mobile agents improve the management efficiency. We compare the performance of two different solutions to gathering MIB-II variables on managed elements: a mobile agent-based one and SNMP. By analyzing response time results, we can say that the mobile agent performs better than the SNMP when the number of managed network elements ranges between two limits: an inferior bound, associated with the number of messages that pass through a large scale network, and a superior bound, related to incremental size of mobile agent.

1 Introduction

Currently, most network management systems use the Simple Network Management Protocol (SNMP) [14] and the Common Management Information Protocol (CMIP) [15], which are based on a centralized paradigm. These protocols use the client-server model, on which the management station acts as a client that provides a user interface to the network manager and interacts with agents, which are servers that manage remote access to local information stored in a Management Information Base (MIB). In these protocols, the operations available to the management station for obtaining access to the MIB are very low level.

Performance management is one of the management functional areas identified in OSI Systems Management and addresses the availability of management information, in order to be able to determine the network load [5]. This management needs access to a large quantity of dynamic network information, which is collected by periodic polling.

The fine grained client-server interaction and the periodic polling generate an intense traffic that overloads the management station, resulting in scalability problems. In this sense, mobile agent is an option to distribute and scale

⋆ Grant holder from CNPq - Brazil.

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 95-102, 2000.

the network management. Mobile agents decentralize processing and control, and, as a consequence, reduce the traffic around the management station, turn asynchronous agent-manager communication (useful when there are unreliable or lossy links), distribute processing load, and increase the flexibility of the management agents' behavior.

Research activities related to mobile code in network management are recent [10]. Magedanz et al. [8] have been one of the first researchers to present issues for the deployment of mobile agents in telecommunications and network management. Management by Delegation (MbD) [7] has been the first paradigm to address decentralization and automation of management tasks by dynamically delegating management functions to agents. Pagurek et al. [4] implement a code mobility infrastructure and a suite of simple tools that interact with agents located on network components. Puliafito et al. [11] use mobile agents to collect information about the state of the network and to perform a micro-management of network devices through multiple variables. Morin et al. [13] implement a management architecture that consists of a set of servers and managers communicating with SNMP agents located on the managed network elements.

The performance of mobile agents in network management is also being investigated. Baldi et al. [2] evaluate the tradeoffs of mobile code design paradigms in network management applications by developing a quantitative model that provides the bandwidth used by traditional and mobile code design of management functionalities. Bohoris et al. [5] present a performance comparison of mobile agents, CORBA, and Java-RMI by using one network element on a ATM network. An array of objects (not real data) is used in order to obtain the response time and bandwidth utilization. Gavalas et al. [6] analyze bandwidth utilization for the SNMP and the mobile agent. Experimental implementation results are presented in terms of bandwidth consumption and response time to obtain an aggregation of multiple variables on a LAN of a few nodes. Sahai and Morin [13] perform measurements of bandwidth utilization of mobile agent and client-server applications on an Ethernet LAN of a few nodes. They also present a single case comparison of response time for the mobile agent and the client-server. Rubinstein and Duarte [12] simulate management tasks performed by mobile agents and SNMP ones, comparing both the approaches, on a topology that consists of a LAN of managed network elements connected to the management station by a bottleneck link.

In this paper, we analyze the scalability of two network management approaches: a mobile agent-based one and one only based on the SNMP. A prototype implementation is evaluated on a topology that consists of a LAN of managed network elements connected to the management station by the Internet. The number of managed network elements is up to 250. Response time results show that the management performance increases when using mobile agents.

This paper is organized as follows. Section 2 presents two implementations of a performance management application: the mobile agent-based one and the one only based on the SNMP. Section 3 describes the methodology of the experimen-

tal study and Sect. 4 reports the obtained results. At last, concluding remarks are presented in Sect. 5.

2 Implementation of a Performance Management Application

We compare two different solutions to gathering MIB-II [9] variables on managed elements: a mobile agent-based one and one only based on the SNMP.

2.1 The Mole Platform and the AdventNet Library

The Mole infrastructure [3] is used in the mobile agent implementation. This system provides the functionality for the agents to move, to communicate with each other, and to interact with the underlying computer system. In Mole, a weak migration scheme is provided, where only data state information is transfered. As a consequence, the programmer is responsible for encoding the agent's relevant execution states in program variables.

Two different types of agents are provided. System agents are usually interface components to resources outside the agent systems. They have more rights than non-system agents (e.g., only system agents can read or write to a file), but they are not able to migrate. User agents are agents that have a "foreigner status" at a location, that means that they are not allowed to do something outside the agent system as long as they can not convince a system agent to give them access to outside resources. Mole uses TCP to transfer mobile agents.

In order to use SNMP functionalities, the AdventNet SNMP library [1] and the snmpd from Linux are also used. The AdventNet SNMP package contains APIs to facilitate the implementation of solutions and products for network management and the snmpd is an SNMP agent that responds to SNMP request packets.

2.2 The Two Solutions

The mobile agent solution (Fig. 1) consists of two agents, one stationary and one mobile. The mobile agent migrates to a network element and communicates by Remote Procedure Call (RPC) with the translator agent that converts the mobile agent request to the SNMP format. This static agent is utilized since in Mole, mobile agents are not allowed to access the resources outside the agent system. The translator agent sends a request to the SNMP agent and obtains the response, which is passed to the mobile agent. Then, the mobile agent goes to the next element and restarts its execution. After finishing its task, the mobile agent returns to the management station.

In the SNMP solution, we have created a fixed agent (the manager) in the Mole infrastructure that sends an SNMP packet to an SNMP agent that responds to this manager (Fig. 2). The manager sends requests to all elements to be managed (one after receiving the response from the other).

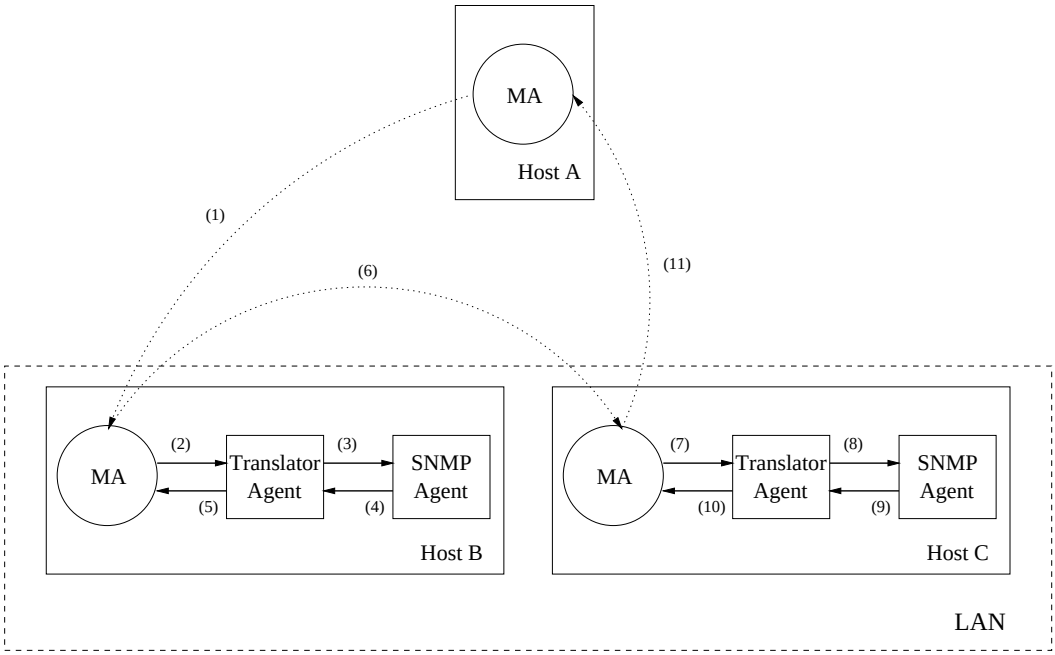


Figure 1. Mobile agent managing two elements.

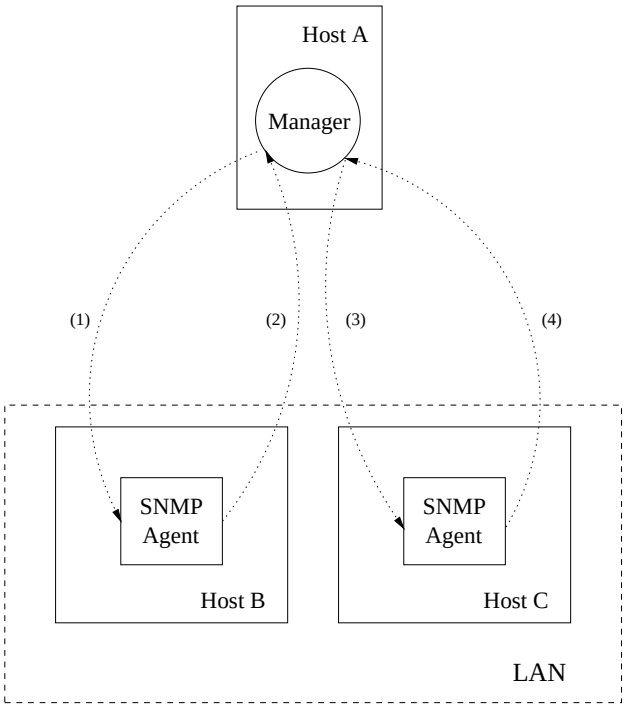


Figure 2. SNMP managing two elements.

3 Experimental Study

We perform an experimental study in order to evaluate the scalability of the two solutions.

The topology consists of two managed network elements (hosts B and C) that belong to a LAN interconnected to the management station (host A) through the Internet (Fig. 1). All the three machines are located in France, two at the PRiSM laboratory and one at the LIP6 laboratory.

In order to evaluate the performance for a great number of managed elements, we alternately repeat the two machines that belong to the LAN, e.g., if we want 5 elements to be managed, we use an itinerary like B, C, B, C, and B.

The considered performance parameters are the average bandwidth consumption on the management station and the average response time in retrieving the MIB-II [9] variable *ifInErrors*, which denotes the number of received packets discarded because of errors, from the elements to be managed.

The tcpdump program is used to measure TCP and UDP packet sizes. All measurements have been performed early in the morning or at night in order to limit the variations of the network performance and to avoid retransmissions of SNMP packets, which would influence the response time results.

Both the solutions have been tested in the same conditions, using the same itinerary. We have made all the tests with the engines (agencies) running uninterruptedly. The number of managed network elements has been varied from 1 to 250 and all the experiments have been repeated ten times.

4 Results

We evaluate here the effect of the number of managed network elements on bandwidth utilization and response time.

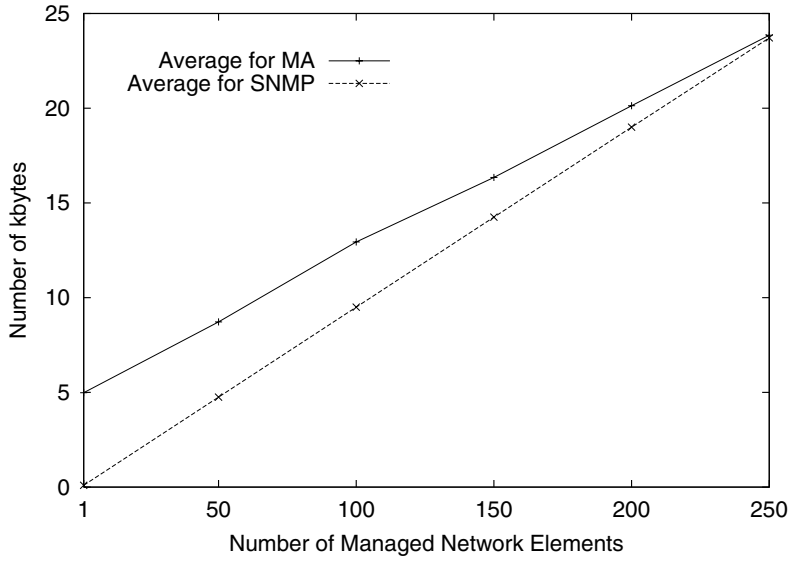


Figure 3. Bandwidth consumption per number of managed elements.

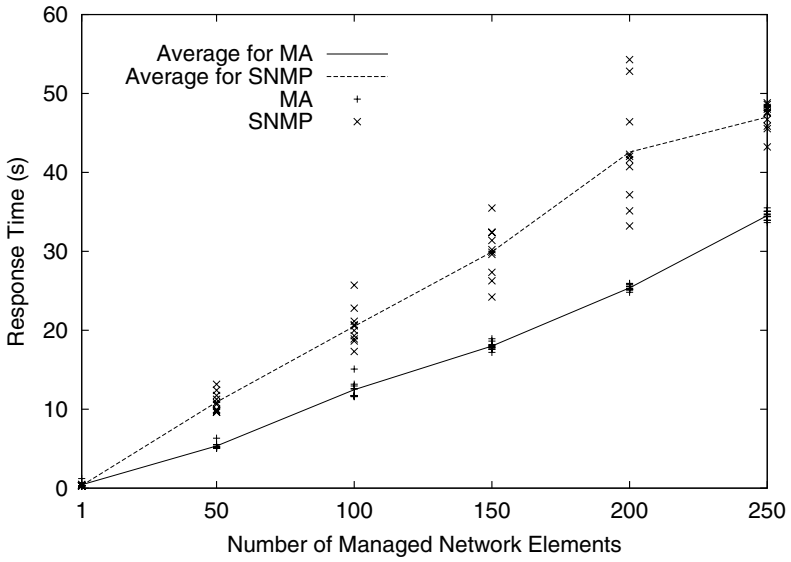


Figure 4. Response time per number of managed elements.

For a small number of managed network elements, the SNMP uses fewer bytes than the mobile agent to perform the task (Figure 3). Nevertheless, as the number of managed elements increases, the overhead associated with several retrievals (PDU GetRequest and UDP's headers) turns mobile agent utilization more suitable, as we can presume by extrapolating the analysis. According to Fig. 3, the initial size of the mobile agent is approximately 2.5 kbps, since it traverses twice the large scale network.

Figure 4 shows that the mobile agent is less sensitive to the network variations (SNMP points are more distributed over time), since SNMP packets traverse oftentimes the large scale network whose performance varies more than the LAN performance. Figure 4 also presents the average response time. For a small number of managed elements, the SNMP performs better than the mobile agent, due to the fact that the SNMP messages are smaller than the initial size of the mobile agent. As the number of managed elements increases, response time for the SNMP grows proportionally, since the time to manage a network element is approximately the same for all network elements. For the mobile agent, response time increases faster when the number of managed elements grows, due to the incremental size of the mobile agent. By extrapolating the analysis, we can conclude that the mobile agent performs better that the SNMP when the number of managed network elements ranges between two limits. The superior bound seems to be high, since it could not be observed in our experiment.

5 Conclusion

This paper has analyzed the scalability of mobile agents in performance management. We have implemented and compared two solutions to gathering MIB-II variables on managed elements: a mobile agent-based one and the pure SNMP.

By analyzing response time results, we can say that the mobile agent performs better than the SNMP when the number of managed network elements ranges between two limits: an inferior bound, associated with the number of messages that pass through the large scale network, and a superior bound, related to incremental size of mobile agent, which turns migration difficult.

Response times for the mobile agent decrease up to 51%, when comparing with using the SNMP.

We conclude that the mobile agent paradigm significantly improves the network management performance when most of the agent movement is inside high bandwidth LANs; in other words, when only a small percentage of the agent movement is inside large scale networks.

Acknowledgments

We would like to thank Pedro Velloso from the GTA for helping with the Mole infrastructure and Luis Costa from the LIP6 for his support during the measurement phase.

References

[1] Advent Network Management Inc. AdventNet SNMP release 2.0. http://www.adventnet.com, 1998.

[2] Baldi, M., and Picco, G. P. Evaluating the tradeoffs of mobile code design paradigms in network management applications. In 20th International Conference on Software Engineering (ICSE'98) (Kyoto, Japan, Apr. 1998), pp. 146–155.

[3] Baumann, J., Hohl, F., Straber, M., and Rothermel, K. Mole - concepts of a mobile agent system. World Wide Web 1, 3 (1998), 123–137.

[4] Bieszczad, A., Pagurek, B., and White, T. Mobile agents for network management. IEEE Communications Surveys 1, 1 (Sept. 1998).

[5] Bohoris, C., Pavlou, G., and Cruickshank, H. Using mobile agents for network performance management. In IEEE/IFIP Network Operations and Management Symposium (NOMS'00) (Honolulu, Hawaii, 2000).

[6] Gavalas, D., Greenwood, D., Ghanbari, M., and O'Mahony, M. Using mobile agents for distributed network performance management. In 3rd International Workshop on Intelligent Agents for Telecommunication Applications (IATA'99) (Stockholm, Sweden, Aug. 1999).

[7] Goldszmidt, G., and Yemini, Y. Delegated agents for network management. IEEE Communications Magazine 36, 3 (Mar. 1998), 66–70.

[8] Magedanz, T., and Eckardt, T. Mobile software agents: A new paradigm for telecommunications management. In IEEE/IFIP Network Operations and Management Symposium (NOMS'96) (Kyoto, Japan, Apr. 1996).

[9] McCloghrie, K., and Rose, M. Management information base for network management of TCP/IP-based internets: MIB-II. RFC 1213, Mar. 1991.

[10] Pham, V. A., and Karmouch, A. Mobile software agents: An overview. IEEE Communications Magazine 36, 7 (July 1998), 26–37.

[11] Puliafito, A., and Tomarchio, O. Using mobile agents to implement flexible network management strategies. Computer Communications 23, 8 (Apr. 2000), 708–719.

[12] Rubinstein, M. G., and Duarte, O. C. M. B. Evaluating tradeoffs of mobile agents in network management. Networking and Information Systems Journal 2, 2 (1999), 237–252.
[13] Sahai, A., and Morin, C. Towards distributed and dynamic network management. In IEEE/IFIP Network Operations and Management Symposium (NOMS'98) (New Orleans, USA, Feb. 1998).
[14] Stallings, W. SNMP and SNMPv2: The infrastructure for network management. IEEE Communications Magazine 36, 3 (Mar. 1998), 37–43.
[15] Yemini, Y. The OSI network management model. IEEE Communications Magazine 31, 5 (May 1993), 20–29.

Visualizing Mobile Agent Executions

Yi Wang and Thomas Kunz

Systems and Computer Engineering
Carleton University
tkunz@sce.carleton.ca

Abstract. Code mobility has the potential to provide more flexible and efficient solutions to traditional distributed applications. However, developing distributed programs with code mobility is quite a challenge and so is the understanding of their dynamic behavior. Graphical visualizations are a promising way to help to understand the dynamic behavior of distributed applications, including those that contain mobile agents. This paper addresses two issues: what needs to be visualized and how do we visualize it. We present an innovative approach to visualizing code mobility in the context of process-time diagrams/message sequence charts. An infrastructure that provides tracing facilities and supports both on-line and postmortem visualization is discussed to demonstrate our approach.

1 Introduction

Several new technologies are evolving for distributed applications. A promising one is based on code mobility, the ability that a chunk of code moves independently from one network component to another. In the remainder of this paper, we will use the term "mobile objects" as generic term for both variants. Generally, there are two forms of mobility implementation: mobile objects and mobile agents. Code mobility affords new opportunities for the distribution of processing and control in the network. There are several areas that may benefit from appropriate use of code mobility:

- If an object needs to exchange a large number of messages with another object in a remote location, they can move closer to each other to reduce network traffic and increase throughput.
- A server program can move services (objects) into a consumer device so that the device can be served even after it is disconnected from the network.
- Code mobility can be used for on-line extensibility of services or software upgrades.
- Server programs can move their services (objects) to a better execution environment to balance the load or to avoid bottleneck and long latency.
- Code mobility is also viewed as an enabling technology to realize the UMTS concept of a virtual home environment (VHE) [5,6].

Code mobility has the potential to provide flexible and efficient solutions and both academy and industry demonstrate increasing interest in this technology. However,

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 103-114, 2000.

developing distributed programs with code mobility is a challenge and so is the understanding of their dynamic behavior. This is due to the fact that those programs involve all the complexity of distributed programs plus problems specific to code migration. Furthermore, the execution of such a program is relatively more complex and non-deterministic. Tools and modeling languages used to understand the static code structure are helpful but they have no explicit way for expressing the distribution of code throughout the network. It is believed that graphical visualization is a promising candidate that can help to gain insight into the dynamic behavior of distributed applications. One supporting argument is that graphics may convey far more information than textual displays. Moreover, graphical visualization uses graphics and animations to present and analyze the program execution instead of static code structure. However, a review of recent publications reveals that no visualization toolkit provides facility for displaying code mobility. With the increasing application of code mobility in distributed network environments, it is essential to have a means to visually describe and illustrate the cooperation, distribution and migration of objects. Application developers can benefit from it during software development and maintenance. It also helps network administrators to monitor those movable objects if they allow them to float inside their network.

Our research addresses two major issues: what needs to be visualized and how do we visualize it. A solution to inserting instrumentation with few modification of source code is provided. An innovative approach to presenting code mobility in the context of a process-time diagram is proposed. An infrastructure that provides tracing facilities and supports both on-line and postmortem visualization is developed.

The research is based on programs using Objectspace Voyager [14] as target environment. The reason for choosing Voyager is that it is an agent enhanced ORB that supports rich code mobility. Another reason is its popularity. Although our target environment is Voyager, the principle developed in this research can be applied to other mobile code/mobile agent platforms.

The paper is organized as follows: Section 2 discusses related work. Section 3 addresses the question of what needs to be visualized. Section 4 focuses on the question of how to visualize, with respect to those interesting events proposed in Section 3. Section 5 overviews the design of the visualization infrastructure. The final section summarizes our contributions and outlines possible areas of future work.

2 Related Work

A collection of articles on the topic of using software visualization to assist parallel and distributed programming can be found in [3,12]. Among existing visualization systems, Polka [17], ParaGraph [7], Poet [2,9,10], and XPVM [11] are discussed in more detail here.

Polka is a 2-D visualization toolkit that provides rich support for concurrent animations. It provides an object-oriented design model to developers. Animations can include any number of graphical views. Semantic zooming is used to resolve scalability problem. With semantic zooming, entire programs and their data set can be presented within one view.

ParaGraph is a performance evaluation and display system that uses a novel approach to collect data. The approach is to instrument the libraries that provide the parallel programming functions. ParaGraph relies on PICL, a Portable Instrumentation Communication Library [8], as the data collection facility and all the information is derived from it. The advantage of this approach is that it minimizes the need for annotation by having the distributed system automatically generate the annotation whenever possible. However, PICL can only provide annotations for the generic communication primitives that it supports.

Poet is a tool that collects and visualizes event traces from applications running in several different target environments, displaying the event occurrences in a process-time diagram. Time in Poet either refers to a notion of physical time (useful for performance debugging) or logical time, capturing Lamport's precedence relation [13] (useful for fault debugging). To manage the complexity of the resulting visualizations for non-trivial executions, Poet supports abstraction facilities in both the process and time dimensions. These abstraction facilities enable Poet to visualize distributed executions on a number of abstraction levels. To achieve target-system independence, Poet makes as few assumptions as possible about characteristics that must be possessed by all target environments. Information describing each target environment is placed in configuration files, allowing a single set of Poet executables to be used for all target environments.

An existing system close to our work is XPVM, an X window based graphical console and monitor for PVM [4]. It provides several different views for monitoring program execution, among which the Space-Time view provides a similar visualization to Poet's process-time diagram. XPVM does not provide instrumentation functionality, nor does it provide a trace library. All the views and functionality in XPVM are driven by information captured via the PVM tracing facility. No annotation of the user program is necessary to use XPVM, as the PVM distribution version 3.3.0 or higher include built-in instrumentation for tracing user applications. Although XPVM is tightly bound with PVM, the mechanism used to drive the views can be applied to other visualization systems.

Polka and ParaGraph both separate event collection and visualization in time. Events are collected at runtime, visualization occurs afterwards (post-mortem). In contrast, Poet and XPVM support both post-mortem and online visualization. In the latter case, visualization occurs concurrently to an executing application. Due to the parallelism and non-determinism involved in event collection, online visualizations tools are usually more complex to implement.

3 What to Visualize

This section will address the question of what needs to be visualized. In other words, what entities, relationships and actions exist in the program that might be portrayed visually. The answer to this question provides a basis upon which the visualization can be built. A brief overview of the target environment, Voyager, will be helpful in understanding why a particular event is selected for visualization.

Voyager is an agent-enhanced ORB, which combines distributed object and agent technology. Therefore, in addition to those capabilities found in most other ORBs, it provides a host of features for supporting code mobility. Voyager is 100% Java™, a concurrent, object-oriented language, and is designed to use the Java language object model [14]. In this section, only those features related to code mobility will be briefly introduced.

Voyager enables an object to be constructed on different hosts through a remote invocation. A proxy object reference is returned so that the local process can use the newly constructed object right away. In Voyager, almost all serializable objects have the potential to move between different locations. Voyager implements mobility in three different ways:

- An object can be treated as a mobile agent.
- An object can be moved explicitly to a new location by obtaining access to the Mobility facet and invoking the MoveTo () operation on that facet.
- An object can be serialized and passed to the remote virtual machine as a parameter of remote invocations on a Voyager object.

The main events of a movable object in its lifecycle can be categorized as follows:

- Creation: Creation is the start point of an object's life. Either a movable object or a stationary object can initiate it. A creation event depicted in the process-time diagram involves two synchronized events from two different trace lines: a creating event on the creator trace and an object-created event on the new trace line.
- Destruction: The garbage collector of JVM will reclaim objects when they are no longer pointed to by any references. Object destruction events are unary.
- Mobility: Mobility is a key characteristic that distinguishes movable objects from other objects. Either a movable object or a stationary object can initiate the mobile process. A movable object with autonomy can also ask itself to move. In general, there are two kinds of events associated with mobility: arrival events and departure events.

There are some other likely interesting events, such as those related to method invocation and threads. These events are not specific to code mobility and quite a bit of research has been done in previous visualization systems [16]. As the goal of our research is to provide a sensible visualization scheme for the purpose of understanding code mobility, our efforts will not focus on these events, but rather on those associated with object creation, destruction, and mobility.

4 How to Visualize

In this section, we attempt to address the question of how to visualize. In other words, ideally, how those interesting events like entities, relationships and actions that exist in the program might be depicted. We first introduce an adaptation of process-time diagrams, which is used in many visualization systems, and then we explain, in more detail, how to apply a similar graphical representation in our visualization.

The graphical representation of the behavior of programs can be carried out either by animation or by process-time diagrams. Animation is good at capturing a sense of

motion and change, and thus may provide an intuitive feeling for the dynamic behavior of the program. However, temporal relationships are difficult to analyze using animation, because the granularity of temporal relationships shown in a single frame of animation is limited. An alternative to animation is the process-time diagram, which presents the execution of the program in a two-dimensional display with time on one axis and individual entities on another. It gives a compact view of the event history, and places much more emphasis on the temporal relationships between entities.

In the progress-time diagram, each entity, such as process, task, and object, is viewed as a sequence of atomic events and represented by horizontal lines, called traces, with time progressing from left to right. Events constitute the lowest level of observable behavior in a distributed execution, such as sending/receiving a message, or the creation/termination of a process. In reality, no global clock exists and events can only be ordered partially. The basic relation between primitive events is the happened-before relation introduced by Lamport [13]. The visualization focuses on interactions between entities instead of the internals of each entity. A symbol, such as an open/solid circle/square, represents important events in each entity. Interactions, such as communication, between entities are shown by arrows, which are drawn from the appropriate event in the sending entity to the one in the receiving entity. In our work, we use the following symbols to represent events:

- Solid squares represent object creation events, while open squares depict destruction events
- Solid circles indicate object-move events
- Solid diamonds represents object-arrive events

The trace lines can be invisible or drawn as solid or dashed according to the state of the entities they represent. Before the creation of an entity and after its destruction, the line is invisible. During the entity's blocked or migration period, the trace line will be dashed. Except for these two cases, the trace line will be solid to depict an active entity. Arrows that depict interactions between objects are vertical for synchronous interaction and sloping for asynchronous one.

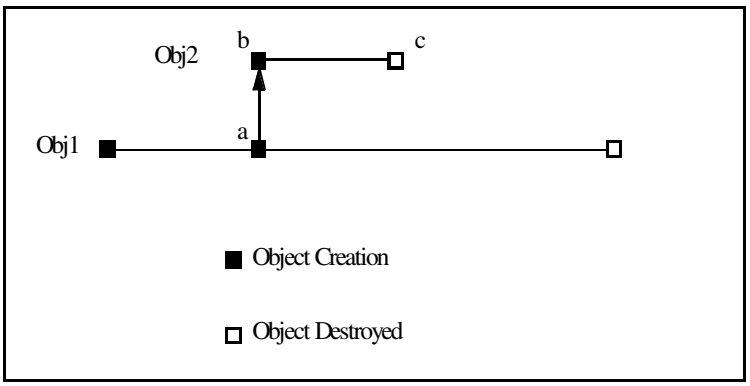


Fig. 1. Object Creation and Destruction

As shown in Figure 1, an object is drawn as a trace line. Object creation involves two synchronized events from two different trace lines: a creating event on the creator trace and an object-created event on the new trace line (with the exception of the very

first object, which, being instantiated by the runtime system, has only a unary creation event). Object destruction is a unary event with no partner.

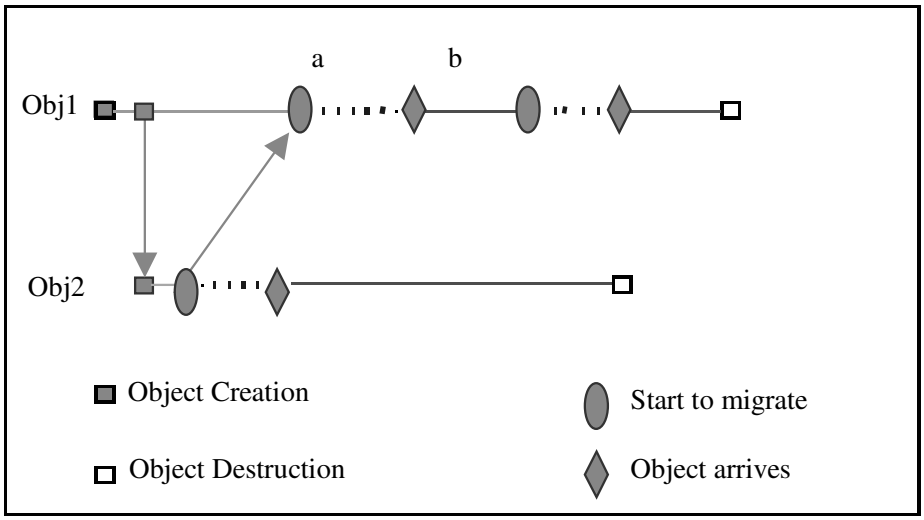


Fig. 2. Object Mobility

While object creation and destruction are relatively simple to visualize, it turned out to be more difficult to identify a proper way to depict code mobility. The solution proposed here is intended to meet the following objectives: the display explicitly indicates the mobility of the objects, minimizes the display space needed, is inherently scalable, and correctly reflects concurrency.

As shown in Figure 2, the trace lines represent objects. It adopts color to encode information about object mobility. The color depicts the change of location. In other words, the trace line will consist of segments with different colors that depict different nodes where the mobile object resides. The color changes when the object arrival event occurs. In this way, mobile objects occupy the same display space as those without mobility and the object migration is explicitly depicted without additional cost. Since the trace lines represent objects, this display has no extra problem in depicting concurrency. Furthermore, we use colors to indicate the change of location, not necessarily to identify the location. In general, if the number of nodes that mobile objects move to is no more than the types of color that the system can provide, we can also represent the nodes using color-coding. If this is not the case, we just change the color of the segment on the trace line alternately to show the migration of the object. We also provide a facility to allow the user to obtain location information by clicking on the individual line segment. Therefore, this type of display will not impose extra limits on the number of objects or object locations it can depict.

5 Tracing and Visualization

A complete visualization system has to implement three components: events need to be created, collected, and visualized.

5.1 Event Generation

Program instrumentation involves the placement of small pieces of code into the operating system, the runtime system, or in the user's source program. The function of this code, referred to as probe, is to report some value to the component responsible for aggregating the event data. In most cases, operating system and runtime environment are almost inaccessible in terms of code insertion. Besides, instrumentation at either of these levels is most unlikely to provide information about abstract, high level events in the application program. In contrast, source code is easy to access and instrumentation at this level provides reliable information about events at different abstract levels. Therefore, source program instrumentation is preferred over the other two and chosen for our system.

In order to gather the expected trace events we need to specify where in the source code the appropriate events are generated, and in most cases, it is necessary to modify the program, either by:

- Modifying source code to generate explicit calls to event log library routines
- Using wrapper classes or method overriding to add tracing to the runtime system API.

These two approaches have their own advantages and disadvantages. A variant of the second approach is applied in our system. Obviously, it is desirable to instrument the source code without or with at least minimal source code modifications. With this goal in mind, we provide an interface *IVisualizable* and a class *Visualizable* that implements the interface. This class contains methods that will be called when an associated event happens. These methods generate corresponding trace events and report them to a local component that is responsible for collecting the data. In the user's program, any movable objects that want to be visualized must inherit from class *Visualizable*. This approach provides a concise solution for both trace event generation and reporting. It allows instrumentation with little source code modification and the programmers do not need to know much detail about the event tracing.

Since Java does not support multiple inheritance, this approach has a potential problem when the movable object class inherits from another class. The Adapter (Wrapper) pattern can be used to solve these problems. We provide a class *VisualProxy* that defines methods with the same implementation as those defined in class *Visualizable*. It serves as an adaptee. Movable objects have to implement the *IVisualizable* interface and compose an instance of the *VisualProxy* class. It acts as an adapter. Clients invoke an operation on an adapter instance. In turn, the adapter calls the adaptee's corresponding operation that carry out the request.

Obviously, a movable object needs a unique identifier. Basically, there are two ways to assign an identifier, a lexical name or a serial number. Serial numbers are more likely to be unique but are almost meaningless to the user. In contrast, lexical names are easier to be understood but it is much more difficult to guarantee their uniqueness. A conjunction of name and serial number can be an obvious solution.

In Voyager, objects execute within runtime environments called Voyager servers. These servers introduce a hierarchical structure. A given computer in a network can host multiple servers; each server holds a number of movable objects. The fact that a host can contain more that one server requires that servers have unique names within

a host. We can use a fully qualified class name to distinguish different classes and an integer number for distinguishing the instances of a given class. Generating identifiers in this hierarchical way, an object could be identified by the conjunction of a hostname, a server name, its class name and an integer. Host and server here refer to the location where an object is created.

5.2 Event Collection

To visualize an execution, events not only need to be created, but also collected and potentially pre-processed. The Event Collection Subsystem is responsible for this task.

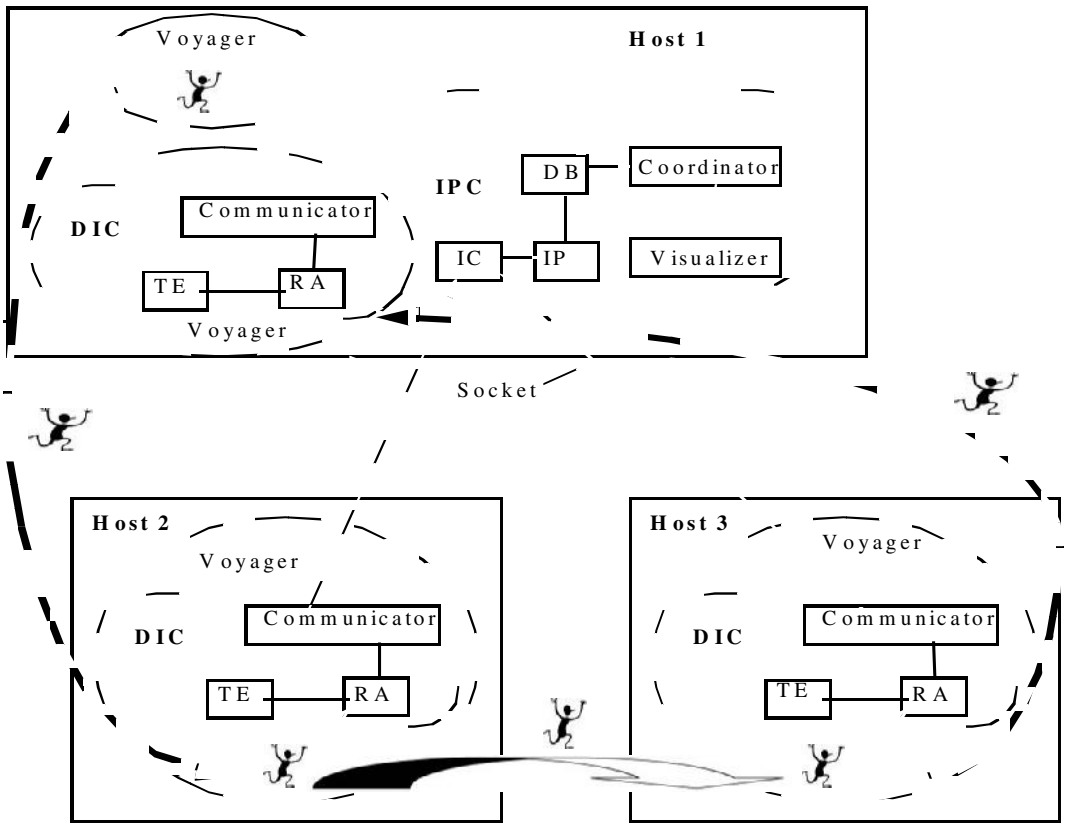


Fig. 3. Event Collection Subsystem

In our system, as shown in Figure 3, there is one central Information Process Center (IPC), typically located on a host that provides visualization functionality. One Distributed Information Collector (DIC) resides on each Voyager server. They are responsible for transferring locally generated trace events to the IPC. When the application is running, the probes capture trace events of interest. They are collected and partially processed by the DICs. These trace event are then fed to the IPC where they are further processed and then either read by the visualization tool for on-line display or stored in a trace data base for postmortem analysis.

This data collection scheme potentially facilitates both on-line and postmortem visualization. On-line visualization is more complicated to implement and may impede the performance of network application. Nevertheless, our system supports this mode for the following reasons:

- Systems that support both on-line and postmortem visualization have a wider usability, for example in performance evaluation and debugging.
- Some visualization activities, due to their nature, must be performed on-line to be effective, such as environment monitoring and interaction/steering.
- Comparing the network bandwidth with the amount of data that will be generated by the tracing system, we believe that trace events will not consume a substantial amount of network bandwidth and thus will not introduce undue overhead.

The event collection subsystem provides a number of services to facilitate event collection:

- Communication Service: The Distributed Information Collector (DIC) and the Information Process Center (IPC) communicate with each other using TCP sockets.
- Encoding and Decoding Services: Since DIC is implemented using Java, while IPC is based on C/C++, encoding and decoding services are provided.
- Description and Interoperation Service: Self-Defining Data Format (SDDF) [1] is chosen as the trace files format because SDDF is highly flexible in representing trace information. SDDF specifies both the structure of data records and data record instances. Therefore, before sending a trace record to the IPC, DIC should send its descriptors first. These descriptors precisely define the content of each trace record and specify the data types and unpacking order of the trace information. IPC interprets each trace record according to its descriptor. The advantage of using this format is that we can add new trace events or rearrange the content of the trace records without modifying the original trace library.
- Time Synchronization Service: This service allows the DIC to adjust its time according to the local time of the IPC. This approach can be used with other mechanism to preserve event causality.
- Event Monitor Service and Timestamp Service: Event Monitor Service and Timestamp Service are used to generate timestamped trace events.

5.3 Event Visualization

The visualization subsystem is based on XPVM [11] and is written using the TCL/TK toolkit [15] with application extended C commands. It supports both on-line and postmortem visualization. After initialization, the TCL interpreter will normally stay in an event loop waiting for X-window events. If the application is idle, two tasks will be scheduled to execute alternatively. One task will read the trace records and save them into a trace file in SDDF format. Another retrieves the trace records from the file and displays them in a process-time diagram. As shown in Figure 4, the visualization system provides the following facilities:

- Trace play control: In addition to "Play", "Fast Forward "and "Stop" buttons, the control panel also provides "Single Step", which allows a single trace event to be

processed, and "Rewind", which resets the visualization system to the beginning of the current trace file.

- Process-Time view queries: More detailed information regarding specific object states can be extracted by clicking on the view area with the mouse button.
- Process-Time view zooming: The Process-Time view can be zoomed in or out to display different perspectives on the same trace data. This facility is provided to control the level of time detail. If a particular area of the view is zoomed in, it will be enlarged horizontally to fill the entire Process-Time canvas area. Scrolling along the time direction is also supported.
- Process-Time view Task Height: In addition to horizontal zooming, the vertical axis of the view can also be scaled. In other words, the height of the horizontal object bars can be shrunk or grown to appropriately utilize the screen area when either a very large or a very small number of objects is displayed. Moreover, the task height need only be adjusted for convenience, since the object's line area can always be seen by scrolling the Process-Time window.
- Customize the display order of traces: By default, the object traces are placed in alphabetic order. Users are also allowed to change the order as they wish by pressing the mouse button and dragging the trace to a designated position.

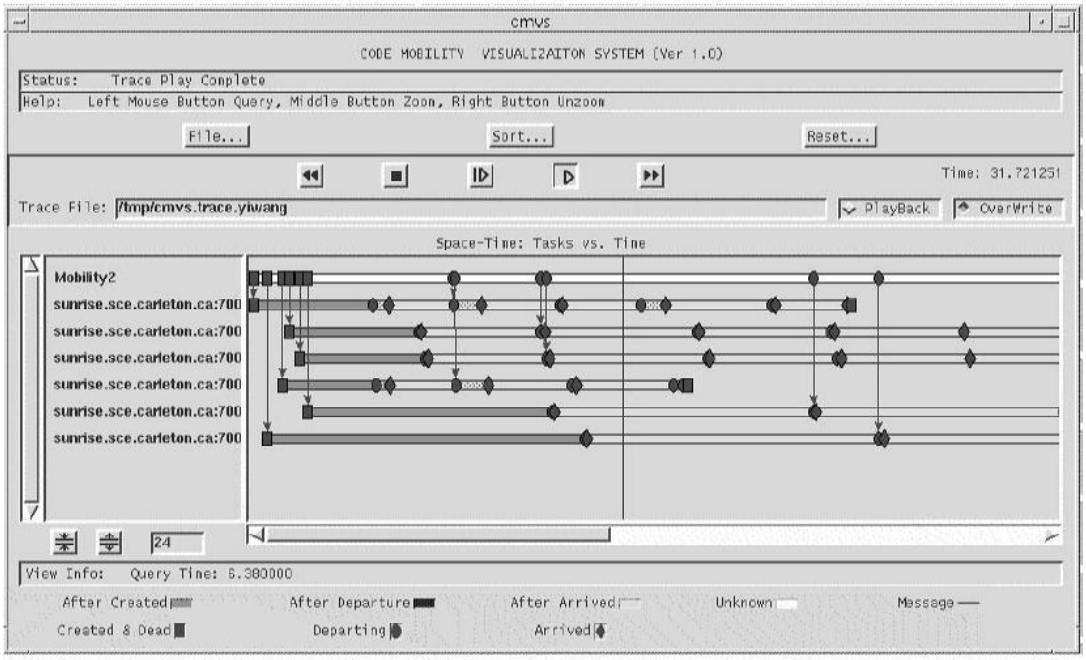


Fig. 4. Visualization Subsystem

6 Summary and Future Work

Program development and maintenance are two of the most vital activities that occur in computing. Each relies on, and can benefit from, an increased level of program understanding. Distributed programs with code mobility involve all the complexity of distributed programs, plus problems specific to object migration. They are

intrinsically complex and hard to understand. With the increasing interest in the application of code mobility, it is important to provide software developers with a tool for understanding those applications during software development and maintenance. It is equally vital to provide network administrators a way to monitor and manage those mobile objects if they allow them to float in their networks. We believe that event-driven visualization can be and will be an invaluable tool for understanding and monitoring distributed programs with code mobility.

In this research, we provide an approach to visually describe and illustrate code mobility by identifying interesting events that could help us understand the execution of distributed programs with code mobility. An information tracing and visualization infrastructure was developed. This infrastructure supports both on-line monitoring and postmortem visualization. It minimizes the need for program annotation by class inheritance or aggregation. It provides a means for visualization with little or no programmer intervention. It facilities mechanisms that allow users to take a closer look of a particular area of the view, and to get more detailed information about a specific event. Although this infrastructure was developed to visualize programs written for the Voyager platform, its principles and structure could be applied other mobile code toolkits.

This research provides a starting point for the following future work:

- Support views driven by other interesting evens such as method invocation and thread interactions. In distributed object-oriented applications, interactions between objects are through method invocations. The objects involved may be located in the same address space or not. Method invocations that involve mobile objects could be helpful for understanding their dynamic behaviors. Multi-threading is a technique that used more and more frequently in distributed applications for its potential performance improvement. It would be desirable to record events related to thread state and synchronization.
- Explore the adaptation of distributed-event collection mechanisms to support the large-scale deployment of mobile agents (many agents, executing in and moving between many locations).
- Deal with the vast amount of information involved in understanding the execution of complex applications. If more event types, such as method invocation or thread synchronization, are intended to be visualized, efforts should be put on how to deal with a huge quantity of trace data. Some means of event abstraction may be needed to support a hierarchy of abstraction levels.
- Develop views driven by dynamic system or network information for system or network awareness.
- Apply the tool to problem of network and distributed systems management, or performance tuning.

Acknowledgements

This work was support by the National Research and Engineering Council, Canada (NSERC) and a research grant by Bell Mobility Cellular, a Canadian wireless service provider.

References

1. Ruth A. Aydt, “The Pablo Self-Defining Data Format”, available at http://www-pablo.cs.uiuc.edu.
2. T. Basten et al., "Vector Time and Causality Among Abstract Events in Distributed Computations", Distributed Computing, Vol. 11, 1997, pages 21-39.
3. T. L. Casavant, “Special Issue on Tools and Methods for Visualization of Parallel Systems and Computation”, Journal of Parallel and Distributed Computing, Vol. 18, No. 2, June 1993.
4. G. A. Geist et al., “PVM: Parallel Virtual Machine, A User’s Guide and Tutorial for Networked Parallel Computing”, The MIT Press, 1994, ISBN 0-262-57108-0.
5. L. Hagen et al., “Impacts of Mobile Agent Technology on Mobile Communication System Evolution”, IEEE Personal Communications, August 1998, pages 56-69.
6. J. Hartmann et al., “Agent Technology for the UMTS VHE Concept”, Proc. of the First ACM International Workshop on Wireless Mobile Multimedia, Dallas, Texas, USA, October 1998, pages 48-57.
7. M. T. Heath and J. A. Etheridge, “Visualizing the Performance of Parallel Programs ”, Technical Report ORNL/TM-11813, Oak Ridge National Laboratory, Oak Ridge, TN, USA, May 1991.
8. M. T. Heath, “Visual Animation of Parallel Algorithms for Matrix Computations”, Proc. of the Hypercube Conference, 1989, pages 735-738.
9. T. Kunz and J. P. Black, "Understanding the Behavior of Distributed Applications Through Reverse Engineering", Distributed Systems Engineering. Vol. 1, 1994, pages 345-353.
10. T. Kunz et al., "POET: Target-System Independent Visualizations of Complex Distributed Application Executions", The Computer Journal, Vol. 40, No. 8, 1997, pages 499-512.
11. J. A. Kohl and G. A. Geist, “XPVM 1.0 User’s Guide”, Technical Report ORNL/TM-12981, Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN, USA, April 1995.
12. E. Kraemer and J. T. Stasko, “The Visualization of Parallel Systems: An Overview”, Journal of Parallel and Distributed Computing, Vol. 18, No. 6, June 1993, pages 105-117.
13. L. Lamport, “Time, clocks, and the ordering of events in a distributed system”, Comm. ACM, Vol. 21, No. 7 (July 1978), pages 558-565.
14. ObjectSpace. Voyager 2.0.0 User Guide, http://www.objectspace.com/Voyager/
15. J.K. Ousterhout, “Tcl and the Tk Toolkit”, Addison-Wesley Publishing Company 1994, ISBN 0-201-63337-X.
16. W. De Paw et al., “Visualizing the Behavior of Object-Oriented Systems”, OOPSLA’93, Washington, D.C., USA, October 1993, pages 326-337.
17. J. T. Stasko and E. Kraemer, “A Methodology for Building Application-Specific Visualizations of Parallel Programs”, Journal of Parallel and Distributed Computing, Vol. 18, 1993, pages 258-264.

Towards Policy-Driven Agent System Development and Management[1]

Larry Korba, Fuhua Lin
National Research Council of Canada
Rm. 286, Building M-50, Montreal Road, Ottawa, Ontario K1A 0R6
Larry.Korba@nrc.ca, Fuhua.Lin@nrc.ca

Abstract A variety of environments currently exist either commercially or in the research community for the development of multi-agent systems for telecommunication applications. Many of them offer the raw tools required for developing agent-based applications. They include provisions for agent mobility, communication, security, and directory facilities among other services. Using a policy-based approach for developing agent systems offers the advantages of systematizing and simplifying development. The approach also offers the potential for dynamic modification of agent system behaviors through changes in policy. This paper briefly reviews some of the previous work in the area of policy management for distributed systems and describes an object-based approach to policy-driven agent development and management.

1 Introduction

In the distributed computing environments found in telecommunications there is a need for adaptability to functional and non-functional application requirements. As well, components used in distributed applications are often hard to reuse in different environments or with different types of services. Often, code must be tailored to specific requirements and peculiarities of the environment.

An approach for dealing with this challenge is to use the notion of policy to express and to modulate the operation of a distributed system in order to deal with these varying requirements. In this work we consider "policy to be *information* which can be used to modify the behavior of a system." [1]. In the case of multi-agent systems, policies may be viewed as the *plans* for multi-agent systems to achieve their goals. Policies may in fact form the fundamental definition of the multi-agent system, describing in detail how the system operates. As well, individual policy elements may be altered to meet changing system needs dynamically.

Considerable work has been done in the development of policy-based distributed management [1][2][3]. Much of this previous work relates to the application of a flexible mechanism for partitioning objects and for describing relationships between objects as related to issues in network management. These principles may equally apply to the design and implementation of any distributed software application.

[1] NRC paper number: 44134

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 115-128, 2000.

Indeed the policy-based approach may be further developed and extended to apply to the development and management of multi-agent systems.

For instance Koch *et al*. develop a multi-level abstraction to deal with management issues from a strategic level to an operational level [2]. At the strategic level the authors use a formal language to define policies and events in order to translate policies into executable rules.

Sloman [1] and Lupu [3] define a policy framework for distributed system management. The framework divides large-scale systems into management domains to provide groupings of objects for policy application and partitioning on the basis of a variety of criteria (geography, object type, responsibility, etc.). In addition the framework describes policies for authorization, obligation, refrain and delegation.

A policy-based framework offers an advantageous approach for developing distributed systems. It can be effective in the management of distributed systems especially in relation to their creation and to their dynamic maintenance. The rest of this paper describes our approach in producing a policy-based framework for agent system development, deployment and maintenance.

The rest of the paper is divided as follows. In the next two sections we describe overall description of the policy-based approach implemented within the Agent Communication and Distribution Environment (ACDE). The section following these four contains specific examples of how policy simplifies agent system development, agent communication monitoring and agent system security in ACDE. Finally, in section 5 we conclude with a discussion and some future research directions.

2 Agent Communication and Distribution Environment

To provide the communication, mobility and security infrastructure required for agent system deployment, we have developed the Agent Communication and Distribution Environment (ACDE) [4]. In this Java-based environment, agents are deployed in a secure fashion, based upon their roles within agent systems. Agent system design involves describing the roles of agents in terms of host, network and agent communication, and security requirements. Agents negotiate for resources as services with agent meeting places (Agent Daemons, see Fig. 1). A security server (trusted third party) mediates overall agent system operation for security responsibilities.

The Agent Daemon is a Java program containing the following components:

- a class loader to remotely load agents as Java classes from a secured directory of a web server
- a multithreaded execution environment for agents,
- services for Agent operation, including directory, security, communication, and
- a proxy for applets. The applet proxy service allows applets running within browsers to operate as agents within agent systems.

Contained within the Agent Daemon is a caching mechanism for agent classes to reduce network overhead for agent class transmission. Once an agent class has been loaded into an Agent Daemon cache, it need not be loaded across the network again, unless its authorization for operation expires. For cached or network loaded agent classes, the Agent Daemon contacts the security server to determine the security

requirements for the agent based upon its role in the agent system. Agent Daemons only allows Agents and Agent Systems with the correct credentials to operate on the network.

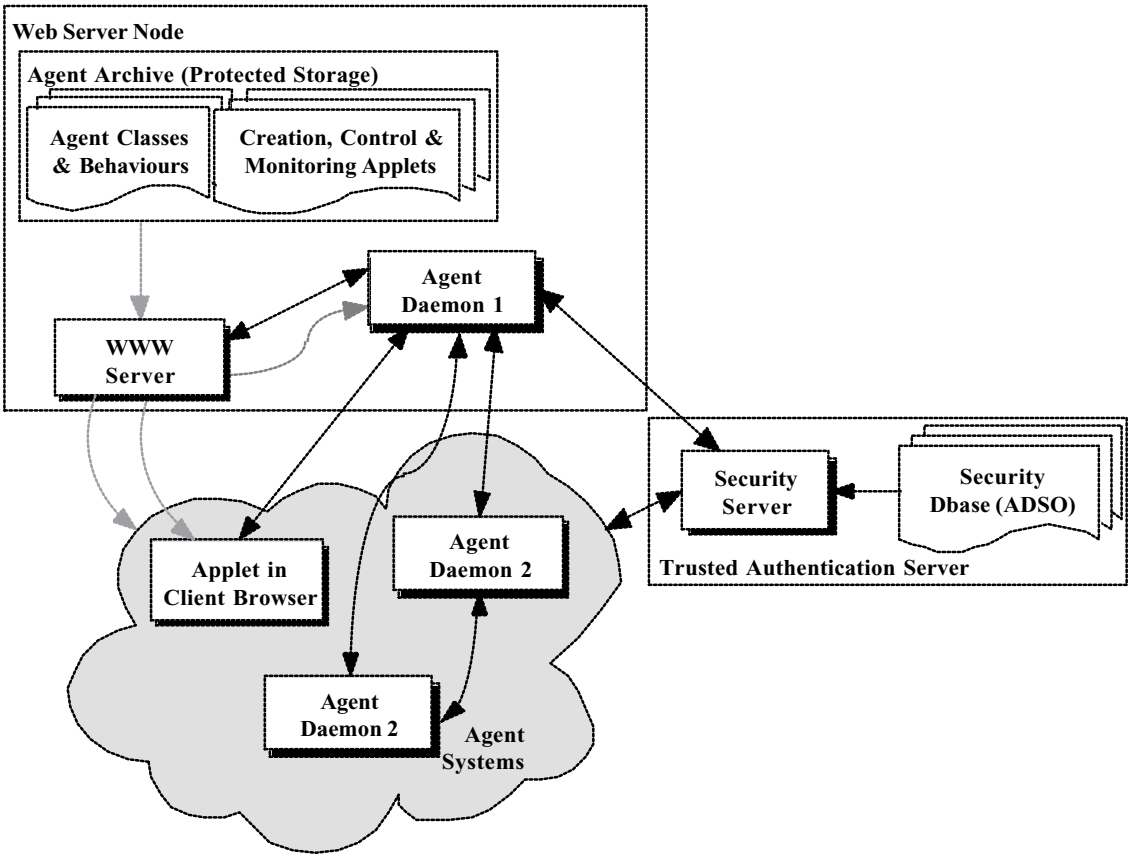


Fig. 1. A simplified diagram of the ACDE. The bi-directional lines indicate agent communication facilitated by the ACDE, while the lighter lines with single arrows indicate agent distribution.

In ACDE policy moderates agent system development, deployment and operation. At the center of the development and operation of agent systems within ACDE is the Agent System Description Object (ASDO). The ASDO contains all policy attributes pertaining to agents and agent systems. The rest of this paper focuses on policy-based approach employed within ACDE.

3 Policy-Based Agent System Environment

Rather than requiring a formal language to specify policy, ACDE stores attributes as member objects of the ASDO. Policy attributes express detailed information in the following categories:

- Identity information (names, certificates related to agents, agent daemons, and other resources and users).

- Distribution requirements (limits for distribution in terms of numbers and locations of agents),
- Communication requirements (type of communication, access requirements, identities of communicators),
- Service requirements (specialized services, APIs required by the agents), and
- Security requirements.

Within ACDE, agent system developers edit the ASDO via graphical user interface. Policy attributes are presented in tabular format with drop-down selection menus allowing a developer to quickly assign policy attributes for an agent system while defining the ASDO. Authorized agents may alter certain policy attributes while an agent system operates. This approach makes it possible to bring more compliance to agent systems as they are used in different environments, where there are different operational constraints. For example, an authorized agent may turn on or off requirements for secure communication, based upon the need to have private conversations. This approach may be used to reduce overall communication latency by not requiring computationally intensive encryption activity for all communications. Since the ASDO contains considerable detailed information about the communication operations and interactions of all agents in the agent system, it is an important element in the monitoring and analysis of agent systems in operation as well (see section 4.4).

3.1 Policy Objects

Policy Objects contain information pertinent to the behavior of an agent system. The approach in ACDE is to simplify creation and maintenance of agent systems through policy while at the same time providing an efficient implementation. Rather than requiring a formal language to specify policy, ACDE handles attributes as member objects of the ASDO. The policy objects are classified into two categories. Each category has a broad range of attributes associated with it.

3.1.1 Policy Categories

The attributes are divided hierarchically into two main categories: agent system and agents. This division associates policy attributes in a logical fashion to either category. Table 1 illustrates the policy categories and the associated policy attributes for agents and agent systems used within ACDE. The list of attributes in Table 1 does shows the details of some of the attributes currently implemented.

3.1.2 Policy Attributes

The attributes of policy objects effectively control particular details of operation for the agents or the agent system as a whole. Effectively, much of the agent system description becomes embodied as attributes within the agent system policy. Beyond their use in agent system creation, some policy attributes may be used to modulate agent system operation. This approach lends itself to object reuse and system adaptability. For instance, an authorized agent may modify policy attributes related to

different aspects of agent or agent system operation to optimize agent system performance. The changes may be made "on-the-fly" with no modifications to the operating code.

Table 1. Attributes of policy contained within the ASDO are organized into different categories.

Agent System Policy	
Category	**Attributes**
System Identification: Attributes identifying the Agent system	Agent System Name
	Creation Date
	Creator: Identity of the creator
	Owner: Identity of the agent system.
	Life time: Elapsed time expected for the agent or agent system to perform its task.
	Public Key
	Agent System Distribution Rules, Agent System Performance Tests
Agent Names: The names of all agents in Agent System	Language: Programming language used to develop agent.
	Deployment Information: Information related to deployment of Agents within Agent Systems.
	Services: Objects related to services required for operation.
	Communication Requirements: Communication details for Agents.
	Agent Distribution Rules, Agent Performance Tests
	Security: Security attributes for the agent.

4 Policy in Application

In this section we describe how policy attributes are used within ACDE. There are four areas: agent system development, agent security, agent system deployment, and agent communication and monitoring. Each of these is described in terms of its application domain in each of the following subsections.

4.1 Agent System Development

Much of the agent and agent system operation is expressed in the ASDO. Through the definition of the ASDO, one develops key characteristics for the agents and the agent system. Agent systems are effectively designed within ACDE via the ASDO. Focussing on the attributes for agent and agent system operation simplifies agent system development. To speed development further, a code generation facility

produces agent shell code based upon the agent policy expressed in the ASDO. Once the ASDO is fully detailed, a single button click generates the Java source files for the agents of the agent system. In addition, ACDE generates the system deployment agent (SDA), an agent that would normally be authorized to deploy an agent system. This aspect is described in section 4.3.1 below.

The code created by ACDE provides the starting point for agent system development. Output from ACDE code generation must be modified to add specific functionality required for agent and system functionality. In the course of development, new policy attributes may be required in support of the added functions. These may be added as policy attributes to the ASDO.

4.2 Tiered Security Policy

In order to do anything useful, a software agent must have access to agent daemon and network resources. On the other hand, access to resources must be restricted to only those processes or agents both authorized and located where there are no other political or systemic restrictions on their use. Security-related attributes contained within the ASDO relate to all aspects of agent operation.

Figure 2 illustrates some of the categories and the attributes of objects associated with secure agent system operation. Attributes often have more than one purpose. For instance, attributes relating to agent communications are also used in the process of negotiation for access to communications resources. Attributes related to security for agents and the agent system also provide important information for agent system deployment, start-up and maintenance.

The security database within the Trusted Authentication Server contains sensitive information needed for authentication exchanges, key distribution, and security audit. While the developer uses the ASDO creation utility to create these security attributes, a security management program may be used to interact with the security server to modify these attributes and the operation of the agent system later.

Table 2. Security attributes associated with individual agents

Agent	Attribute
A1	a1, a2, a3
A2	a1, a4
A3	a2
A4	a3, a4

ACDE also uses a two-tiered security policy approach. The first tier involves the agent perspective regarding its requirements for operation. The second tier involves the agent daemon and its policies regarding allowed access to resources by agents operating on its network location.

For an agent to be deployed at an agent daemon within ACDE, the security policy attributes for the agent must match with the allowable agent policies for the agent daemon. The situation is illustrated in Table 1. Security attributes (a1-a5) form part of the policy definition for the agents. Similarly, the agent operating environment

within the agent daemon has control policies for access to resources (Table 2). When an agent arrives at an agent daemon, it must present its credentials and its policy attributes. If the credentials are in order and the Agent Daemon has matching access control policies for each of the agent's policy attributes, then the agent will be granted access to the resources.

Table 3. Access Control Policies associated with Agent Daemons. Presence of a policy **p**n (n is an integer) indicates acceptance of an agent with security attribute **a**n.

Agent Daemon	Access Control Policies
AD1	p1, p2, p3
AD2	p1, p4, p5, p8
AD3	p1, p2, p3, p4, p5,
AD4	p1, p2, p3, p4, p5, p6, p7, p8

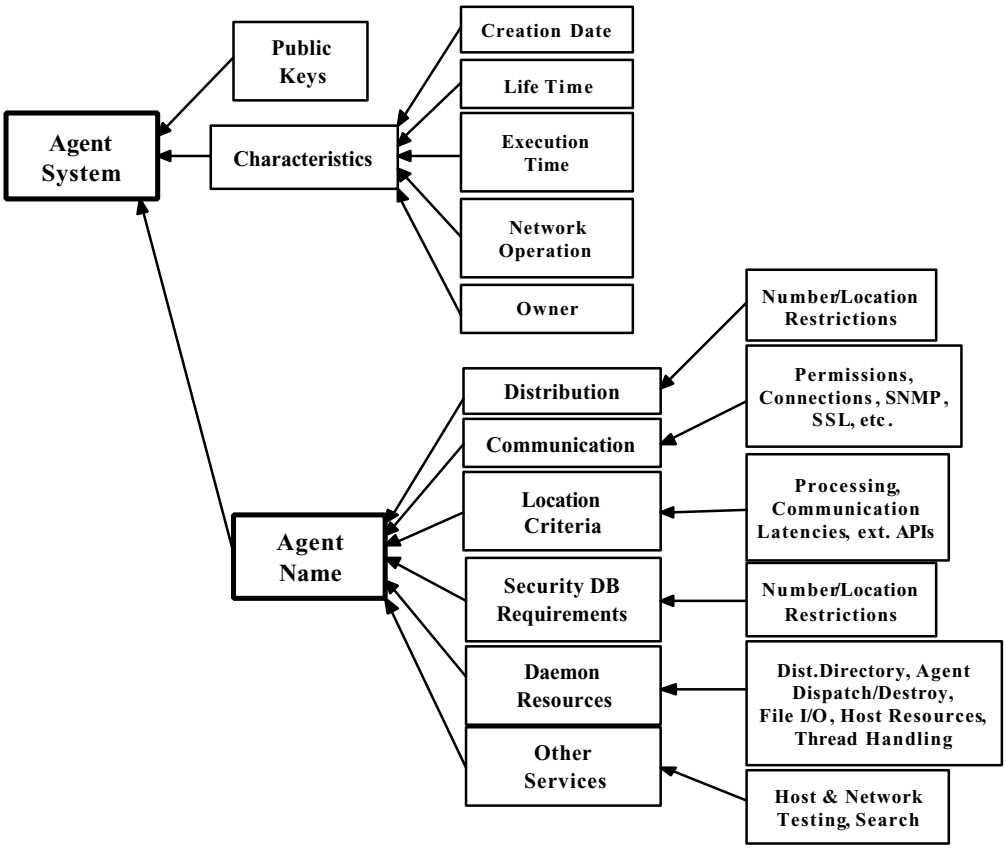


Fig. 2. Some of the ASDO attributes associated with the agent system and agents.

In the example illustrated in Tables 1 and 2, Agent A3 with security attribute a2 would only be operable in Agent Daemons that have security policy p2 (all Agent Daemons but AD2). In addition to specifying security control policies for Agent Daemons and security policy attributes for agents, there is a migration policy relating where an agent may be allowed to operate within a network. The migration policy indicates preferred and restricted nodes for agent migration. One node may be

preferred over another due to its proximity to a particular resource, for instance, a telephony application program interface. An Agent Daemon may be restricted from access by a software agent due to its limited computing resources or the preferences of the computer user.

4.3 Agent System Deployment

While a distributed, agent-based approach offers the potential advantages of problem partitioning and application scalability, one of the challenges when implementing such systems relates to optimization through appropriate agent deployment. It is often important to optimize various aspects of system operation. Within an agent system, correct location of agents may improve performance characteristics of an agent in particular, and/or for the agent system as a whole. Depending on the application, different operational requirements may be optimized. For instance, the security of operation of an electronic commerce application may be optimized by appropriate location of agent elements with respect to physical network elements, e.g. firewalls, databases containing sensitive information, etc. Agents requiring protected information available on a particular computer located within a demilitarized zone may be distributed there in order to access the information locally rather than over the network as a security requirement. Transaction latency is another performance characteristic especially critical for telecommunication and electronic commerce applications. Strategic agent location may reduce overall transaction latency. Compromises between the disparate system requirements are necessary to meet apparently conflicting goals. For instance, some techniques for reducing transaction latencies may introduce security vulnerabilities.

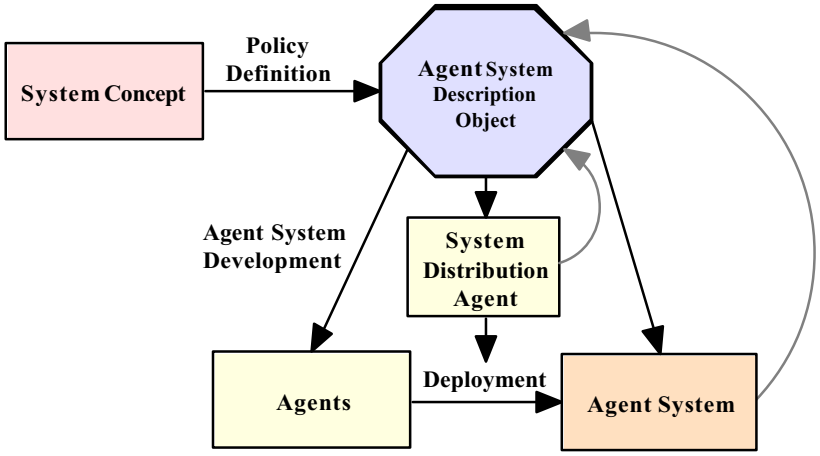


Fig. 3. This diagram illustrates the role of the ASDO within for system development and deployment and operation within ACDE.

Within ACDE, the policy expressed within the ASDO governs agent system deployment. Figure 3 illustrates the pivotal role the ASDO plays in the overall operation of the environment.

4.3.1 System Deployment Agent

With ACDE, starting up an agent system involves the network movement and instantiation of agents at different agent daemons operating on different network nodes, the execution of performance tests and their analyses to determine appropriate agent locations and the distribution of information regarding agent location and agent communication connections. In our approach, to simplify matters, a single entity, called the System Distribution Agent (SDA) orchestrates agent system distribution, configuration and overall management. Once again, key to the deployment and operation of agents using ACDE is the ASDO [5]. Included in the ASDO are attributes that are actually Java objects containing methods for performing some of the tests required to determine optimal agent placement for agents.

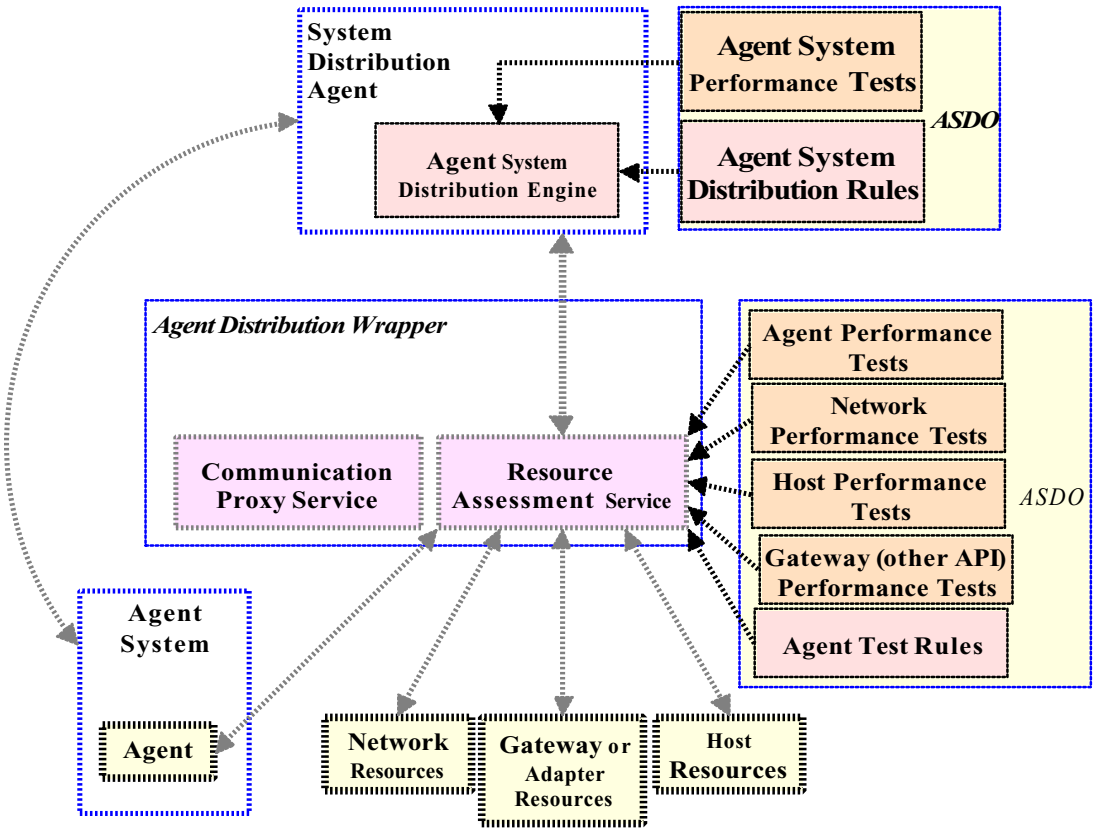


Fig. 4. Schematic diagram illustrating the operation of the Agent Distribution Wrapper.

Our approach equips every agent of the Agent System with an Agent Distribution Wrapper. The Agent Distribution Wrapper (see Fig. 4) has three functions:

1) It provides mobility for the agent.
2) It provides a resource assessment service for the SDA. Using a separate thread of execution from the agent, it tests the performance of the agent and the performance of any resources required by the agent (network, host, or the communication gateway). It communicates with the SDA on a command-response basis or on an event basis whenever the combined results of performance tests indicate a performance bottleneck.

3) It provides a communication proxy service for handling communication transitions during agent redeployment.

The SDA handles initial agent system deployment and management of agent system performance using the elements of the ASDO that relate to agent deployment. Agents may be deployed to network locations specified ahead of time, or they may be deployed on the basis of the competition of performance tests. The tests are contained within the agent wrapper generated by ACDE. The tests are activated by remote method invocation by the SDA. Typical tests include:

1) Agent Performance Tests: tests of the performance of an agent while it is operating within an agent system. How responsive is the agent to communications with its peers or parents with the current agent network configuration.
2) Network Performance Test: tests how well the agent accesses a particular network element (or set of network elements) from an Agent Daemon location.
3) Host Performance Test: determines the performance of the Agent Daemon host for certain types of operations (disk access, floating point operations).
4) Gateway Performance Test: On the basis of agent location, this test determines the performance that can be expected for accessing gateways with which the agent must interact.

Host and network performance tests form a generic type of test. A library of these tests is available when constructing the agent wrapper to simplify the development process.

4.4 Agent Communication and Monitoring

The communication services included within the ACDE offer a type of "back door" communication monitoring. To be allowed to operate within an Agent Daemon, agents must maintain objects reflecting the identity and nature of communications with its published class. An Agent Daemon may access information within the Agent regarding recent data exchanges. The information includes: contents of communication streams, identities of entities connecting for remote invocation, and multicast events requested and received. In addition to this information offered via the communication infrastructure, the ASDO provides details of the communication connection information for agent systems. In combination, this arrangement offers more detailed communication monitoring than would otherwise be possible using extensions of Java's Security Manager class alone. An agent may be monitored for the type of communications (via Java's Security Manager), as well as the context and content of the communication streams.

The Agent Daemon of the ACDE monitors agent communications using a reflective monitoring subclass of the ACDE communication class library. The Agent Daemon maintains a reflective communication monitoring service to make monitored communication streams available to authorized monitoring agents or applications. Authorized agents or applets may remotely enable the reflective communication

monitoring service for all agents of an agent system. The reflective communication monitoring service monitors both unicast and multicast communications. The service makes the monitored data available on an event basis for the monitoring agent. In the situation illustrated in Fig. 5, the unicast communications between Agent A and Agent B are monitored using the communication monitoring service of Agent Daemon 1 and 2. A communication monitoring agent, operating at Agent Daemon 3 registers to receive events from the communication monitoring service of Agent Daemon 1 and 2. Whenever Agent A attempts a remote method invocation with Agent B, the communication monitoring service operating in Agent Daemon 2 makes the details (invoking class, objects exchanged, Internet Protocol addresses of each agent) of communication events available to all registered monitoring entities. A similar same chain of events occurs when Agent B attempts a remote method invocation of Agent A.

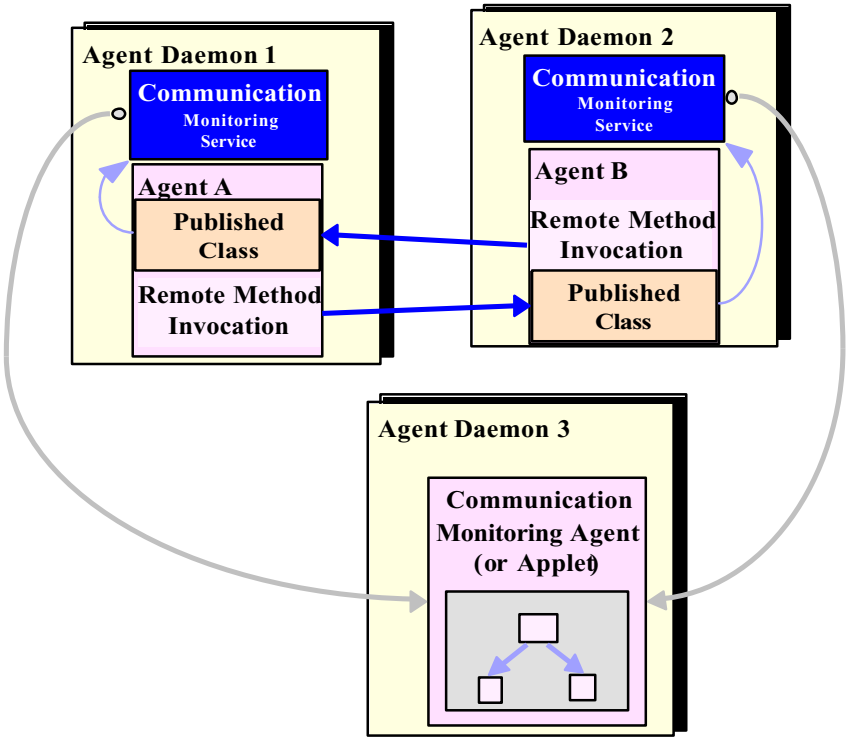


Fig. 5. This diagram illustrates the reflective unicast communication monitoring implemented in ACDE. A communication monitoring service monitors both unicast and multicast communication streams. In this case, a monitoring agent displays a logical view of agents operating on the network.

To give an agent system orientation to agent communications, ACDE embodies an agent system viewer. The viewer presents a graph-based representation of a selected agent system operating on a network. The viewer provides a logical view of agent systems operating on a network. Each edge of an agent system graph represents a communication link while nodes may represent agents or services used by the agents.

Agent communication monitoring offers the ability to discover agent systems operating within a network. A monitoring agent registers with all of the Agent

Daemons in a network. By means of monitoring all communications, the monitoring agent builds a logical diagram indicating the "communication structure" for the agent system. This approach produces a snapshot of all agents or agent systems operating at any time.

Agent system policy as expressed in the ASDO plays an important role in communication monitoring. Attributes within the ASDO relate detailed information about the different communication resources required for the operation of every agent. The communication-monitoring agent uses this information to determine the relationship between agents operating within the network. Fig. 6 illustrates the graph view of agent systems presented by ACDE. In this example, there is only one agent daemon running on the network at IP address 132.246.128.203. There are two agent systems operating, and a single agent controlX. AgentA1 and AgentB1 are two instances of one agent type, with a peer-to-peer communication relationship. They both communicate with the agent daemon where they are located to access resources. PingMasterControl1 and PingBot1 form another agent system in operation on the network. The object that looks like a speaker connecting these two agents indicates a multicast object communication. PingMasterControl1 sends messages to command PingBot1 (or other PingBot agents running on other agent daemons) via multicast

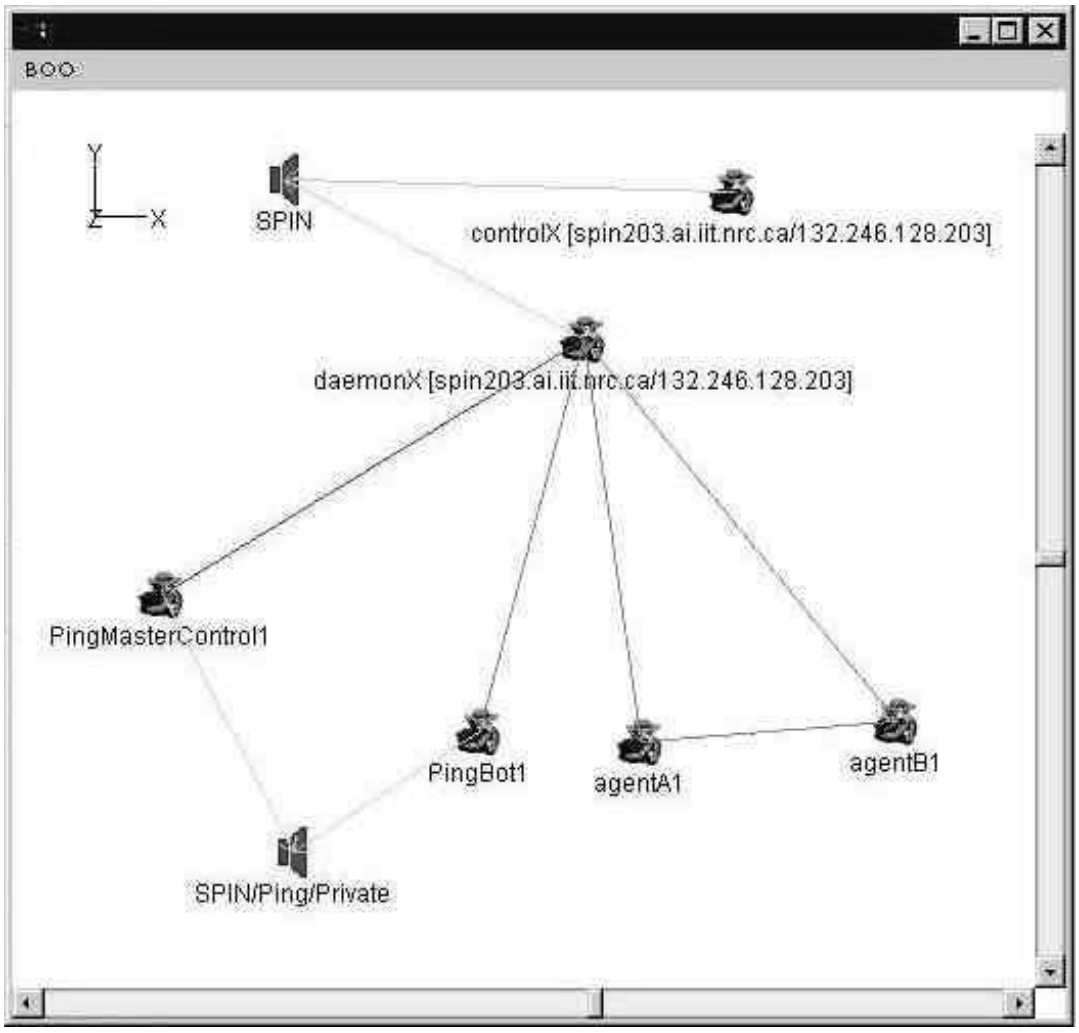


Fig. 6. Screen snap shot showing an agent system in operation using ACDE.

topic SPIN/Ping/Private. The agent controlX is a special agent used to manage the operation of any agent operating at any agent daemon on the network. It uses the SPIN multicast topic to discover the operating agent daemons in the process of creating a distributed directory.

5. Discussion and Conclusions

There are a number of Java-based mobile agent environments available now [6]. While these packages tend to offer far more than required with respect to agent mobility, in many cases they offer fewer security provisions and little or no provisions for monitoring of agent communications. In this paper we have described the agent distribution and communication environment (ACDE). Through the use of an object containing categorized agent and agent systems policy attributes, ACDE offers some benefits in the area of security definition and development, agent system deployment, agent system development and agent communication monitoring.

One benefit implemented in ACDE is rapid development through automated code generation. As well, policy attributes define many different aspects of agent and agent system operation, including deployment, communication and security requirements. While the automatically generated code must be modified to provide application-specific agent functionality, the process effectively speeds agent system development. To simplify agent system deployment, the automatic code generating facility also uses policy attributes to generate a System Deployment Agent. In ACDE, the ASDO contains elements that modulate agent and agent system operation. Authorized agents may alter certain policy attributes modulating system performance in an interactive manner. For example, an agent may turn on or off its requirement for secure communication, based upon the need for private conversations. This approach reduces overall communication latency by not requiring computationally intensive encryption activity for all communications. Moreover, authorized ACDE agents use the policy object to monitor agent communication. Communication monitoring allows agent systems to be discovered using the relationships described in the ASDO, the directory listings for agents at agent daemons and the monitored communications between agents and other objects.

Further work involves extending the policy framework to include more policy attributes including those for the management of ontology, and other inter-agent or resource interactions. As well, automatic generation of the ASDO through agent system monitoring could prove useful for monitoring and discovery of foreign, unknown agent systems operating on a network. Further work includes the investigation of distributing policy as an approach for handling some of the scalability issues normally associated with centralized policy management techniques [6].

In addition, we are investigating alternatives for dealing with scalability of the management of communication between agents. This research relates to the use of communication managers and Colored Petri Nets to model and to develop high-level agent conversations [8].

References

1. M. Sloman. Policy driven management for distributed systems, J. Network and Systems Management, vol. 2, no. 4, pp. 333-360, Plenum Press, 1994.
2. T. Koch, C. Krell, B. Kramer. Policy definition language for automated management of distributed system, In Proc. Second IEEE Int. Workshop Systems Management, pp. 55-64, Toronto, Canada, June, 1996.
3. E. Lupu. "A Role-Based Framework for Distributed Systems Management," PhD. Dissertation, Imperial College, Dept. of Computing, London, July, 1998.
4. L. Korba. Towards Securing Network Management Agent Distribution and Communication, In Proc. Sixth IFIP/IEEE Int. Symp. on Integrated Network Management (IM'99), Boston, MA, May 24-29, 1999.
5. L. Korba, R. Liscano. A Distribution Framework for a Messaging System, In Proc. First Int. Workshop on Mobile Agents for Telecommunication Applications (MATA'99), Ottawa, ON, October 6-8, 1999.
6. Links to other mobile agent tools collected by AgentBuilder. available via WWW at URL: http://www.agentbuilder.com/AgentTools/ .
7. M. Blaze, J. Feigenbaum, J. Lacy. *Decentralized Trust Management*. In *Proc. of the 17th Symposium on Security and Privacy*, pp. 164-173. IEE Computer Society Press, Los Alamitos, 1996.
8. F. Lin, L. Korba, incorporating Communication Monitoring and Control Facility in Multi-Agent Systems, IEEE Conference on Systems, Man, and Cybernetics, 8-11 October 2000, Music City Sheraton, Nashville, Tennessee, USA.

Modeling an OMG-MASIF Compliant Mobile Agent Platform with the RM-ODP Engineering Language

Florin MUSCUTARIU(*+) and Marie-Pierre GERVAIS(*)

* Laboratoire d'Informatique de Paris 6 (*), 8, rue du Capitaine Scott — F 75015 PARIS
+ CS TELECOM (+) 28, rue de la Redoute — F 92263 FONTENAY-AUX-ROSES Cedex
{Florin.Muscutariu, Marie-Pierre.Gervais}@lip6.fr

Abstract. In order to model telecommunications services as mobile agent system, we are defining a methodology based on the RM-ODP standards. Our approach makes the distinction between the service behavior specification, that is independent of the support environment, and the complete service specification that must take into account the target environment. To obtain this specification, the designer must be able to model the target environment according to the concepts used in the methodology, i.e., the RM-ODP concepts. We describe in this paper such a modeling activity. The target environment that we consider is an OMG-MASIF compliant mobile agent platform. We model it by using the RM-ODP engineering language.

Keywords: methodology for agent-based services development, mobile agent architecture, RM-ODP, OMG MASIF standard

1. Introduction

We will present in this paper a model of an OMG MASIF-compliant platform using the RM-ODP engineering language. This work is integrated in the ODAC[1] project developed at LIP6.

The ODAC project aims to provide a methodology for designing telecommunications agent based services using the Reference Model of Open Distributed Processing (RM-ODP)[6]. The methodology provides methods, models and tools to define, in a first step, the so-called "behavioral specification" of a telecommunications service, that is the description of a telecommunications service behavior as a set of interacting agents[2], i.e., a multi-agent system [2, 3]. This specification is independent of the

[1] ODAC stands for Open Distributed Applications Construction

[2] An agent in this paper is considered as a software entity that acts autonomously on the behalf of another entity (human, software or organization) [1].

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 129-137, 2000.

support environment. The second step in the methodology is the targeting, the projection of the behavioral specification on an operational environment in order to define the so-called "operational specification". It must provide the telecommunications service designer with tools, first for modeling the target environment, and then for representing the service specification according to this environment model. Modeling the target environment must be according to the concepts used in the methodology, i.e., the RM-ODP concepts.

This paper is concentrating on this modeling activity. The target environment we consider is an OMG-MASIF compliant mobile agent platform, such as Grasshopper[3] [4, 5]. We focus on the engineering viewpoint of the RM-ODP, which deals with the mechanisms and functions required for the support of distributed interactions between ODP-objects in the system. Thus we provide the model of such a mobile agent platform by using these RM-ODP engineering concepts.

2. ODP Concerns

RM-ODP is a standard developed by ISO and ITU-T that defines an architectural framework within which support of distribution, internetworking and portability can be integrated in order to specify a distributed processing system [6]. The specification of a complete system is divided in viewpoints relevant to some particular area of concern during the design of the system. There are five viewpoints:

1. *Enterprise viewpoint*: a viewpoint on the system and its environment that focuses on the purpose, scope and policies for the system;
2. *Information viewpoint*: a viewpoint on the system and its environment that focuses on the information semantics and information processing performed;
3. *Computational viewpoint*: a viewpoint on the system and its environment that enables distribution through a functional decomposition of the system into objects which interact at interfaces;
4. *Engineering viewpoint*: a viewpoint on the system and its environment that focuses on the mechanisms and functions required to support distributed interactions between objects in the system;
5. *Technology viewpoint*: a viewpoint on the system and its environment that focuses on the technology choices for that system.

The computational viewpoint is concerned with “when” and “why” objects interact, while the engineering viewpoint is concerned with “how” they interact. In RM-ODP, an ODP *object* is defined as a model of an entity. A computational object is the basic entity resulting from the decomposition of the problem done previously during the specification process. The ODP objects involved in an engineering specification are called *engineering objects* (EO). A *basic engineering object* (BEO) is the engineering object that requires the support of a distributed infrastructure. It is the direct mapping

[3] Grasshopper is a mobile agent platform developed by IKV++ GmbH (www.ikv.de)

of a computational object. We consider that a basic engineering object (BEO) may be an agent.

The engineering language, used in the engineering viewpoint, defines the concepts of cluster, capsule, node, nucleus and channel. *A cluster* is a configuration of basic engineering objects forming a single unit for the purposes of deactivation, checkpointing, reactivation, recovery and migration. A *capsule* is a configuration of engineering objects forming a single unit for the purpose of encapsulation of processing and storage. A *node* is a configuration of engineering objects forming a single unit for the purpose of location in space, and which embodies a set of processing, storage and communication functions. The node is under the control of a *nucleus*, which is responsible for creating groups of engineering objects, for making the communication facilities available and for providing other services.

A set of nodes could form an engineering interface reference management domain (IRMD). The IRMD is a naming domain and determines the policy for content, allocation, tracking and validation for the engineering interface references (EIR). The EIR is an identifier for an engineering object interface available for distributed bindings. A *channel* is a configuration of engineering objects providing a binding among a set of interfaces to basic engineering objects, through which interactions can occur. There is a relation of inclusion illustrated in the Fig.1.

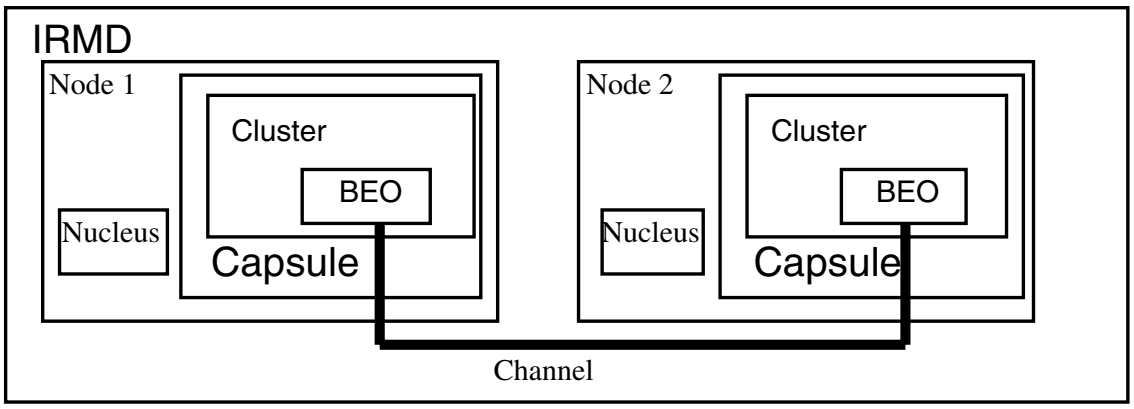


Fig.1. Hierarchical organization of the engineering objects

3. The OMG-MASIF Architecture

MASIF is about interoperability between agent systems [4]. The areas of MASIF contributions contain agent management, agent transfer, agent and agent system names, agent system types and location syntax. The architecture of a MASIF compliant distributed agent environment is composed of agents (stationary and mobile), agencies, places and regions. The *Agency* is the runtime environment, while the *place* provides a logical group of functionality inside an agency. A *Region Registry* maintains information about the components associated with a different region. The *Region* may exist to facilitate the management of the distributed components. The Core

Agency represents the functionality required by an agency for agent execution support (Fig2).

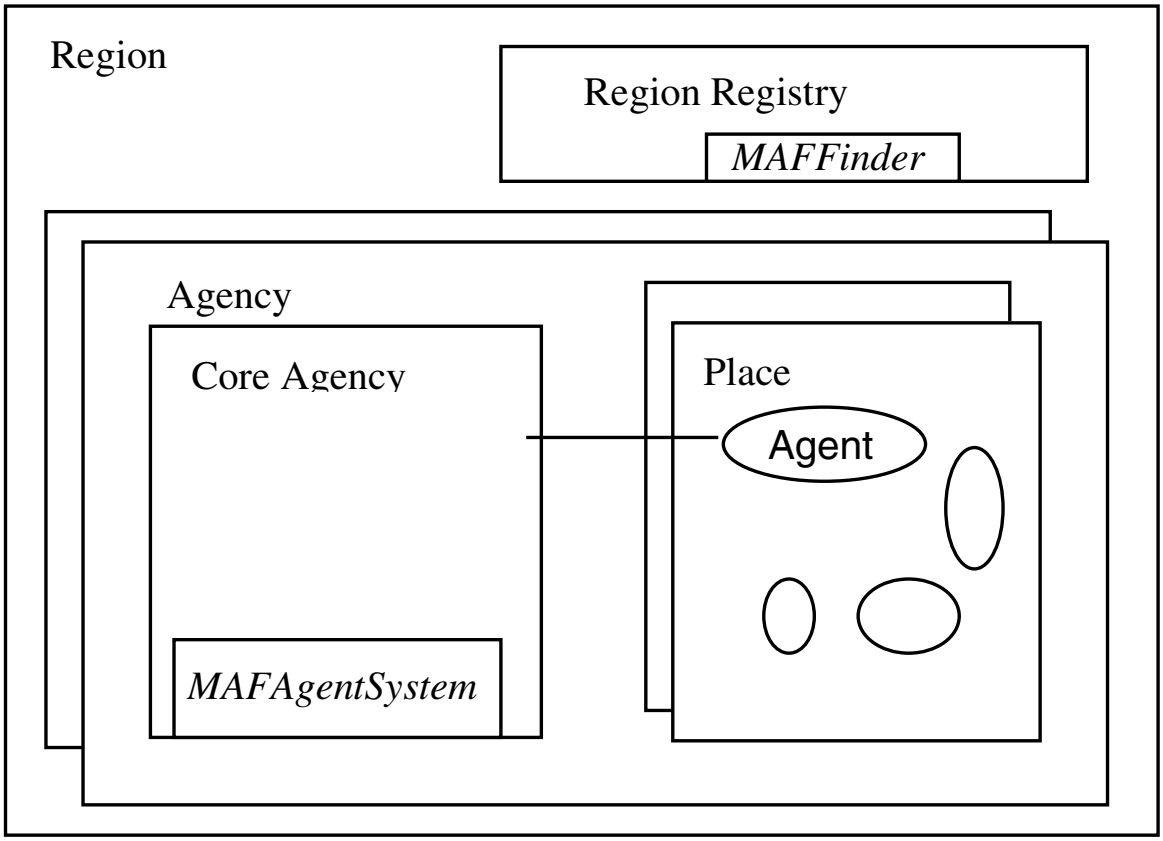


Fig. 2 Hierarchical Component Structure

MASIF is a collection of definitions and interfaces that provides an interoperable interface for mobile agent systems. Two interfaces are defined:

- MAFAgentSystem interface;
- MAFFinder interface.

4. Specification of a MASIF Platform Using the RM-ODP Engineering Language

An engineering specification defines the infrastructure required to support functional distribution of an ODP system by: a) identifying the ODP functions necessary to manage physical distribution, communication, processing and storage; b) identifying the roles of different engineering objects supporting the ODP functions.

In order to do this, we specify:

1. *A configuration of engineering objects*, structured as clusters, capsules and nodes;
2. *The activities* that occur within those engineering objects;
3. *The interactions* of the engineering objects.

For achieving that, we respect the engineering language rules such as: interface reference rules, binding rules, cluster, capsule and node rules, etc.

4.1. Engineering Objects Configuration

The implementation of an ODP *node* in a MASIF platform is an *agency*, and all the agents in an agency share the common processing, storage and communication. A node, implemented as an agency, is a member of an *interface reference management domain (ODP-IRMD)*, implemented as a *Region*. The *nucleus*, implemented as a c*ore agency*, provides a set of node management interfaces for each capsule within the node. The implementation of a *capsule* is a *place*. As we said before, the representation of a BEO may be an agent. If so, the agent autonomy makes us consider that we must have only one BEO per cluster. Actually, since an agent may migrate and initiate autonomous behavior, this reduces the number of agents per cluster to one. If the BEO is not represented as an agent but as a normal object, we may consider several BEOs in a cluster as an option.

According to the ODP rule establishing that an ODP-IRMD contains a set of ODP nodes, a region contains a set of agent systems (Table 1). The region must provide unambiguous object references in the naming context that correspond to the ODP-engineering interface references. This is done by implementing the MAFFinder naming service, which is an IDL interface defined in MASIF. The Agent System, viewed as an ODP-Node in the engineering viewpoint, provides a set of management methods and objects to support agent location and management tasks. These management methods are grouped in an ODP specification in the Nucleus. An agent, viewed as a BEO in the ODP specification, is located in a place. An ODP capsule represents a MASIF place.

MASIF Architecture	*ODP Engineering Viewpoint Architecture Description*
Region MAFFinder	Interface Reference Management Domain (IRMD)
Core Agency MAFAgentSystem	Nucleus

Agent System	Node
Place	Capsule
Agent or Object	BEO

Table 1. ODP engineering concepts vs. OMG-MASIF concepts

Agent communication is outside the scope of the MAF specification and is done by the implemented system communication infrastructure, e.g., CORBA object communication, RMI or RPC. Modeling these implementation choices would result in a channel specification in the ODP engineering viewpoint.

4.2. Engineering Objects Activities

The functions of an agent system are:

- Transferring an agent;
- Creating an agent;
- Providing globally unique agent names and locations;
- Supporting the concept of a region;
- Ensuring a secure environment for agent operations.

We specify these functions of an Agent System with the ODP functions.

The coordination of an agent transfer, which corresponds to the migration management of a single BEO, is done by the coordination migration function. This uses the cluster management function and the capsule management function delegated to the node level.

To create a BEO, the cluster management function is implemented at the node level. The node management function assigns an engineering interface reference within a nominated engineering interface reference management domain (IRMD). The IRMD information is maintained by the engineering interface reference tracking function. The security functions are responsible for the security policies established by a security authority in a security domain. The correspondences are shown in Table 2.

OMG MASIF	*ODP Functions*
Transferring an Agent	Coordination Migration Function
Creating an Agent	Cluster Management Function
Providing Globally Unique Agent Names and Locations	Node Management Function
Supporting The Concept of a Region	Engineering Interface Reference Tracking Function
Ensuring a Secure Environment for Agent Operations	Security Function

Table 2. ODP functions of an Agent System

The appendix details the classification of the MAFAgentSystem and the MAFFinder interfaces methods as ODP functions.

4.3. Engineering Objects Interactions

MASIF addresses three types of interactions related to interoperability:

- Remote agent creation;
- Interaction needed for the agent transfer;
- Agent method invocation.

A client could be a non-agent program or an agent from an agent system having the same system type as the destination agent system or not. This client authenticates itself to the destination agent system and interacts with the destination agent system to request the creation of an agent. In an ODP specification, a BEO interacts with the Node for the creation of another BEO.

When an agent transfers to another agent system, the agent system creates a travel request providing information that identifies the destination place. In order to fulfill the travel request, the destination agent system transfers the agent's state, authority, security credential and the code. In ODP language, we say that the destination Node transfers the BEO and the data associated.

An agent invokes a method of another agent or object if it has the authorization and a reference to the object. This reference can be obtained from the MAFFinder. In the ODP specification, this is described as an interaction between a BEO and the IRMD.

4. Conclusions

The standardization process of the mobile agent environment reduces the necessary effort for target environment modeling. In this way, most of the standard target environments can be modeled using the corresponding standard. Depending on the implementation choices made for non-standard aspects, the full modeling process could be completed with particular mobile environment solutions.

Providing a model for different target environments is our goal in order to achieve "operational specification" at the end of the specification process, passing through the methodology that we are developing in the ODAC project.

The work in progress done by the international research community to standardize the mobile agent platforms, to unify and to merge the existing standards, permits a better efficiency in the methodology development corresponding to the actual or future platforms that are implementing these standards [7].

References

[1] "Mobile Software Agents: An Overview", VU Anh Pham, Ahmed Karmouch, IEEE Communication Surveys, 1999

[2] "Enhancing Telecommunication Service Engineering with Mobile Agent Technology and Formal Methods", Marie-Pierre Gervais, Alioune Diagne, IEEE Communications Magazine, July 1998

[3] " Specifying and Verifying the Behavior of Telecommunications Services " by J.F. Dauchez and M.P. Gervais,, in Proceedings of the 6th International Conference on Intelligence in Networks (ICIN'2000), Arcachon, France, January 2000

[4] The OMG Mobile Agent System Interoperability Facility (MASIF) Specification, http://www.omg.org/cgi-bin/doc?orbos/97-10-05;

[5] "Basic and Concepts", Grasshopper Development System, IKV++.

[6] Open Distributed Processing – Reference Model, ISO 10746/ITU-T X.90x

[7] OMG Agent Technology Green Paper, Agent Working Group, 1 March 2000

Appendix:

MAFAgentSystem methods	*RM-ODP Functions*
Create_agent	Cluster Management Function
Fetch_class	Related to RM-ODP technology viewpoint
Find_nearby_agent_system_of_profile	Trading Function
Get_agent_status	Cluster Management Function
Get_agent_system_info	Node Management
Get_authinfo	Authentication Security Function
GetMAFFinder	Node Management
List_all_agents	Trading Function
List_all_agents_of_authority	Trading Function
List_all_places	Trading Function
Receive_agent	Migration Function
Resume_agent	Cluster Management Function
Suspend_agent	Cluster Management Function
Terminate_agent	Cluster Management Function
Terminate_agent_system	Node Management Function

Appendix 1. The MAFAgentSystem methods described as RM-ODP functions implemented at the Node level.

MAFFinder methods	*RM-ODP Functions*
Register_agent	Engineering Interface Reference Tracking Function
Register_agent_system	
Register_place	
Lookup_agent	Trading Function
Lookup_agent_system	
Lookup_place	
Unregister_agent	Engineering Interface Reference Tracking Function
Unregister_agent_system	
Unregister_place	

Appendix 2. The MAFFinder methods described as RM-ODP functions at the IRMD level.

Active Networks for IPv6 Communication Redirection

Mouhamadou Lamine Diagne – Thomas Noel – Jean-Jacques Pansiot

LSIIT – Pôle API – Boulevard Sébastien Brant – 67400 Illkirch-Graffenstaden
Tel: (33) 03 88 65 55 03 – Fax: (33) 03 88 65 55 01

Abstract. In this article, we propose an Active Network mechanism based on IPv6 to forward communications. Indeed, with the use of the Internet Protocol improvements in its version 6, especially the concepts introduced in IPv6 mobility, we can use active nodes to perform the redirection of the packets of a unicast and UDP communication that we choose in advance. This redirection can be done transparently of one of the correspondents of this communication.

1. Introduction

The objective of this article is to apply active network technology to introduce a new concept, which is *communication mobility*. This is the capacity to move one communication from a host to another. Note that this is different from usual mobility where all communications are moved with a host. We assume that there is at least one active node between the initial communication correspondents.

In this paper, we limit our research to unicast communications. For instance, we assume that we are receiving a video transmission on our personal workstation and we want to redirect it toward a host connected to a video-projector. With this mechanism the communication is not closed but it'll be re-routed on a second host. We notice that in some cases this host may not be able to receive the same data formats than the previous one.

A solution is to do the redirection by acting over the distant application if the station supports format conversion. The first communication is stopped and another one intended for the new target will be established. We notice that this solution is tedious and costs too much time. It can be used only if the sending host is able to convert data formats, and if we are able to remotely control the distant application.

An alternative to this is to use the network to demand to the source to redirect the communication. If it is able to it, the communication between this host and the host chosen will be normal. Otherwise, it'll be done at an active node transparently to the source itself. The active node will load the conversion code. With this method, we gain time because it won't be necessary to contact the source and to stop the communication. Another advantage is that it'll be possible to initiate the

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 139-150, 2000.

communication redirection not only from the receptor but also from the host chosen and even from a host, which is not involved in the communication.
This concept of communication movement is similar to already existing mechanisms in telecommunications networks. For example with telephone networks we are able to redirect:

-A call destined to a station towards another one for example during our absence.
-A current communication when a person who puts a communication through to a colleague.

In these cases, the correspondent doesn't have to stop his communication and make a second call.

In the following we limit ourselves to the search of an active node and the communication redirection. The active network technology used is not detailed in this paper and the mechanism to load the conversion code will be studied in a future work.
We'll call Corr1 and Corr2 the initial correspondents, Corr3 the station replacing Corr2 in the redirected communication, and Initiator the initiator of the communication displacement (i.e. the station which demands the displacement of the communication for Corr3) which can be represented by Corr2, Corr3 or a third station not involved in the communication.

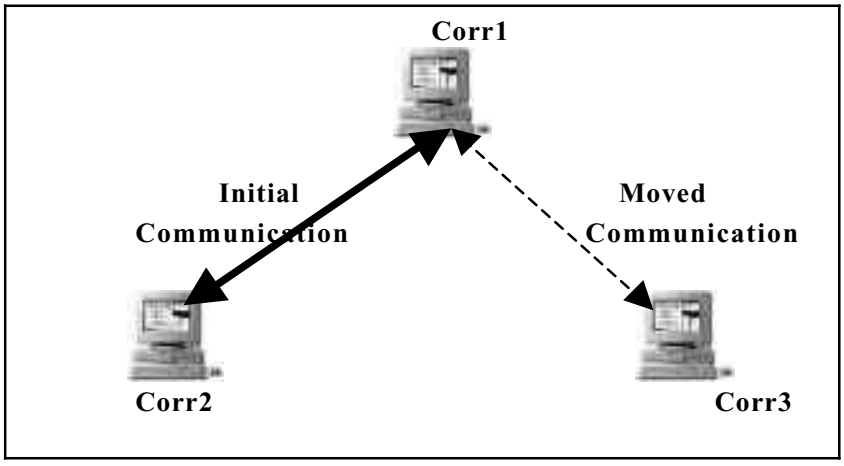


Fig. 1. General set up of the movement of a communication

We are looking for a generic system independent of the applications. It's trivial that all application won't be adapted to use the displacement of the communications. For example, we are not going to displace a file transfer because it's not conceivable to have a part of the file on a station and the other on a second one.
Generally, the communications to displace will be communications without persistence. Principally, these communications are done with the user like a videoconference, IP telephone, or some interactive applications.

In the following paragraph, we'll talk about the related works. In paragraph 3 we'll give an overview of IPv6 [6] protocol before showing the different procedures of the

communication mobility in paragraph 4. In the rest of the article we'll present the different messages, the communication between Corr1 and Corr3, the initiator of the displacement and the security. We'll finish with the conclusion and the future work.

2. Related Work

As we said it, communication mobility is a new concept that we'll implement. However, some existent works such as the XMove [12] pseudoserver and the terminal mobility with IPv6 are related to the subject.

2.1. XMove Pseudoserver

Xmove is a tools implemented as a pseudoserver which exists on Unix operating systems. It allows the displacement of X11 applications from a server to another one. This command allows a graphical interface of an application executed on a station to be displayed on another server of the same station or on a second one. It's a transfer of X data, which is strongly linked, to the server features. Furthermore, it increases network load.

2.2. Terminal Mobility

In this paragraph, we'll talk about the terminal mobility with the IPv6 protocol. When a mobile node is in its home network, it receives the packets destined to it like any stationary host. When it moves to another network then, with IPv6 mobility, its packets are sent to it directly if the correspondent node has an entry in its cache for this mobile or via its home agent which is its representative in its home network. To put it briefly, when a mobile node moves from a network to another all its communications move with it.

2.3. Conclusion

The principle of communication displacement is a mechanism that is not proposed yet. Indeed, there's no question of displacing a graphical interface or all the communications of a station but one or more of its communications. Furthermore, this displacement is more generic since independent of the existent interfaces and the operating systems on the stations. To this end, we have to know existing communications, which sets the problem of their identification. This will be detailed in paragraph 4.1.
Like in terminal mobility with IPv6 mobility [8] where a care-of-address change creates no modification in the transport and application layers, the communication displacement also is performed transparently to the applications of the stations. It's explained by the fact that to redirect a communication we essentially use the improvements already present in the IPv6 protocol.

3. An Overview of IPv6

The IPv6 protocol was created to fill in the gaps of IPv4 and to allow moving away from the threats on the evolution of Internet. With IPv6, we see in particular an extension of the address capacity, a simplification of the format of the header, an improved support of options and extensions, a better management of mobility and security.
Contrary to IPv4, in this new protocol options are not included in the IPv6 header but rather in separated headers representing the extensions, which can be placed between the IPv6 header and the upper layer header. Each header is identified in the precedent by the next header field.
The principal extensions are the following:

- *Hop-by-Hop extension*: every node along a packet's delivery path must examine it. However, it's possible that a node do not know the option type. Then its behavior depends of the two high-order bits of the type field:

  -00: the node skips over the option and continues processing the header.
  -01: the node discards the packet.
  -10: the node discards the packet and returns an inaccessibility ICMPv6 message.
  -11: the node discards the packet and returns an inaccessibility ICMPv6 message if the destination address is not multicast.

- *Destination extension*: there are two types of extension destination. The first is examined by the first destination, which is in the destination address field of the IPv6 header, and by all those listed in the routing header. The second type of destination extension is only examined by the final destination.

- *Routing extension*: it imposes the passage through a list of particular nodes and the last one in this list is the final destination. The segment left field of this extension defines the number of remaining nodes to visit and only these nodes are authorized to decrement the value of this field.

- *Fragment extension*: it is used by the source to give information on the fragmentation of the packets. With IPv6, only the source can fragment packets; the routers are not authorized to it contrary to IPv4.

The mobility in IPv6 is better supported with the elimination of the triangular routing of IPv4. That is, a mobile node can send a *binding update message* to be joined directly by a correspondent, avoiding passing through its home agent. This mobile node uses its *care-of-address* as source address in the IPv6 header of its packets to allow them to go normally through the routers. The mobile node's home address is transported in the *home address* option.

4. Communication Mobility Procedures

We define a communication as a flow of information between two applications. To effect the redirection of a communication, we consider several steps:

-The identification of the existing communications among which we want to redirect one or more. To do this, we have to get information on all the current communications.
-The transfer of the information about the communications towards the host which effects the redirection.
-The last step is the redirection of the selected communication. In this step, the packets, which were destined to the first host, will be received by the host towards which we have demanded the redirection.

4.1. Identification of the Communications

To allow us to know the existing transmissions, we'll place on every station an agent, which we call a Host-Agent (HA), which registers the different communications of the host. The scope of each HA is limited to the host on which it is placed. It perpetually listens at a particular port. This concept is similar to the mechanism used in the service location protocol (SLP) [10]. The HA is able to send requests and responses to another HA or a network node.
The record of a communication contains at least the following data:

- The correspondent's address
- The type of the communication: multicast or unicast (so far, we limit our research to unicast communications)
- The flow label
- The port numbers of the source and the destination
- The transport protocol used (only UDP is examined so far)
- We could insert a communication type (video code, audio, etc).

To identify the IPv6 packets which belongs to the communication, a node could use only the source address and the flow label, which is unique for each source.
However, certain members of the IETF such as Cisco suggest to guarantee the unity of the flow label only on a link; the router has a correspondence table and depending of the input link and the flow label value this table allows to determine the output link and the new flow label value. If this proposition is accepted then it'll be very impossible to identify the packets with only the source address and flow label. In this case we'll use source and destination addresses and ports, and the protocol. The router will have to examine five fields instead of two fields with the first solution: they are the five fields examined by IPv4 routers to create a context.

4.2. Discussion between Concerned Nodes

To displace a communication of Corr2 will require exchanges of messages between Corr2 and Corr3 after it's HA has determined all the current communications. Indeed, to demand the redirection of the packets of a communication, Corr3 must have all the information allowing identifying it. That, it'll demand them to Corr2 which must respond; we assume that before the demand, Corr3 knows Corr2's address.
Having all necessary information, it is now possible for Corr3 to choose the communication to displace and to demand its redirection.

4.3. Displacement of the Communication

After choosing the communication to displace, Corr3 demands its redirection by using information on it. It'll send this demand to an active node not known beforehand and placed between Corr1 and Corr2 (both included).

The movement request, after it is accepted, will allow Corr1 and Corr3 to communicate.

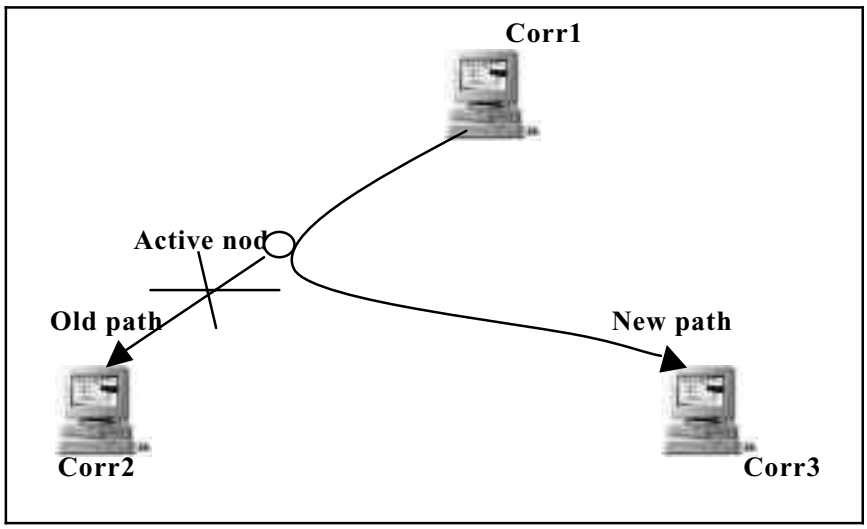


Fig. 2. Movement of a communication

5. Different Messages

To displace a communication from a host to another, we have to effect some exchanges of messages between the concerned nodes. First, the messages involve the Host-agents to discover the existent communications and secondly certain network nodes will be implicated.
The principle messages we propose are the following:

- The *comm-request* message (communication request message) is sent to a HA by another HA to ask for information on the current communications. In this message, the HA specifies the communication type(s) needed and the protocol used in it (unicast or multicast transmission, protocol TCP or UDP: so far only unicast transmissions with UDP protocol are studied).

- The *comm-reply* message (communication reply message) is also sent by a HA to answer to a *comm-request*. This response contains the features of the transmissions (with the type and the protocol demanded) that the host is ready to give. This message allows to know all the communications, which can be displaced, and to choose one.

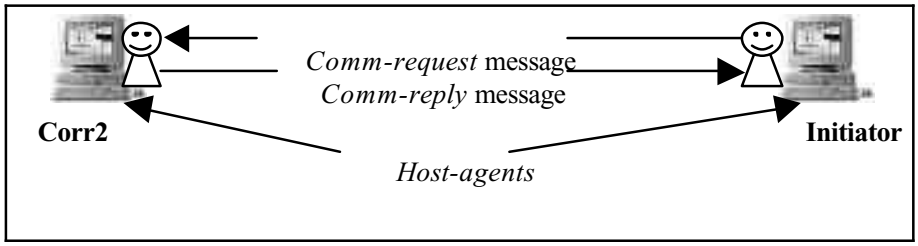


Fig. 3. Discussion of the Host-Agents

- The *comm-recept* request message (communication reception request message) is sent by the HA of the initiator of the displacement (if different of Corr3) to the HA of Corr3. This message contains the description of the communication to displace and the address of Corr2. Corr3 responds by sending a *comm-recept-reply* message in which it accepts or not to receive the communication. If the response is positive, Corr3 will have to achieve the actual displacement mechanism by emitting a *comm-choice* message.

- The *comm-choice* message is sent by the HA of Corr3 to inform to the HA of Corr2 about the communication chosen. This message must be acknowledged with a *comm-choice-acknowledgement*.

- The *move-request* message specifying a communication to displace is sent towards an active node placed between Corr1 and Corr2 and is acknowledged by emitting a *move-ack* message. The node, which undertakes the redirection of the packets of the communication and possibly of the conversion of the data formats by loading the conversion code (not studied in this article) is not known in advance. The mechanism used to determine this node and to redirect the communication is detailed in the following paragraph.

Having this set of messages, it is now possible to displace a communication of Corr2 and to allow Corr1 and Corr3 to communicate.

6. Communication between Corr1 and Corr3

After Corr3 has received from Corr2's host-agent the necessary information on the communication to move and before requesting the redirection of the packets, it has to launch the adequate application to receive the communication and to send a *comm-choice* message to Corr2 to notify it of the choice of the communication. Corr3 will issue this message until the reception of an acknowledgement.
We notice that after sending a *comm-choice* acknowledgement, if it receives a packet of the communication, Corr2 will redirect it towards Corr3 by using an encapsulation.
Now that Corr3 is ready to receive the communication, it requests its movement by sending a *move-request* message.

6.1. Movement Request of a Communication

The packets of a communication can be redirected only by nodes, which handle them i.e. the communication's correspondents and routers on the unicast route between these correspondents. Then, we have to address the request to them.

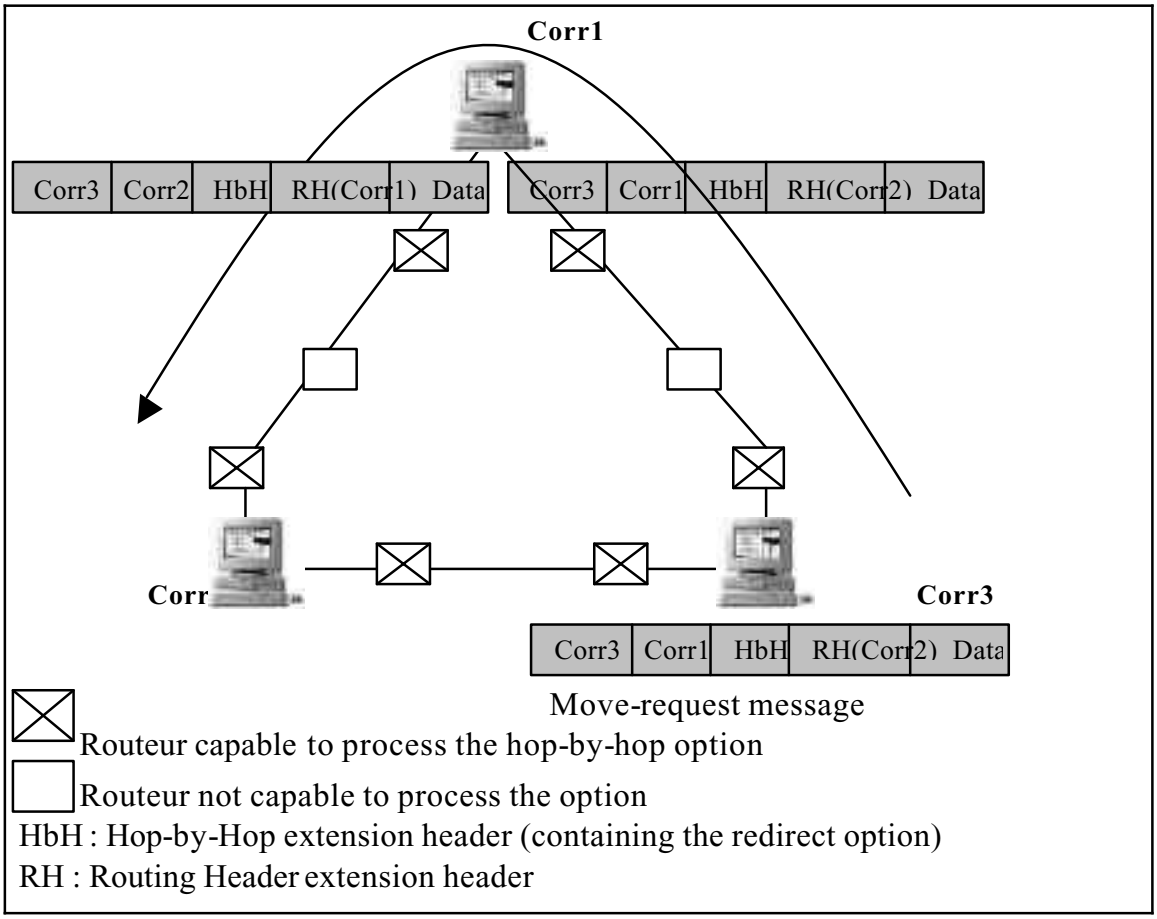


Fig. 4. Movement request of a communication

For this end, we define a new IPv6 hop-by-hop option which we call "*redirect option*". It contains the source and destination addresses and ports, and protocol used by this communication.
That is, we'll use an IPv6 packet with hop-by-hop and routing extension headers to send the *move-request* message. In the hop-by-hop extension header we'll put the "*redirect option*" defined above and in the routing extension header we'll place the address of Corr2; then, the initial value of the "segment left" field will be "1". The address of Corr1 will be in the destination address field of the IPv6 header. This mechanism is used to reach all the nodes between Corr1 and Corr2. Then, when the *move-request* message arrives at a node:

-If it doesn't know the redirect option type, it must ignore the option and forward the packet.
-If it knows it (the node is active), and if it is placed between Corr3 and Corr1 (i.e. the segment left field value is "1") it forwards the packet without any action.
-If it knows the redirect option and is placed between Corr1 and Corr2 (i.e. the segment left field value is "0", it is decreased by Corr1) it has to effect the conversion and the redirection of the communication's packets. It discards the packet and sends an acknowledgement to Corr3.

With this mechanism, the packet will reach Corr1 without any modification. If Corr1 supports the format conversion and communication redirection, it loads the required code and redirects the communication towards Corr3. Otherwise, it decreases the "segment left" field value and permutes the addresses in the destination address field and in the routing header before forwarding the packet. Then, it will be the first active node reached which will effect the data conversion and the redirection of the communication. Note that it is the routing header that allows determining if the packet has already reached Corr1.

6.2 Redirection of the Communication's Packets

The node having to effect the redirection encapsulates the packets of the communication to move and forwards them towards Corr3.
Ours mechanism would decrease in performance if the same node is chosen for the redirection of a large number of communications. However it scales well if many communication are redirected by different nodes.

Corr2 keeps a table where each entry corresponds to a moved communication. An entry is kept as long as Corr2 receives update messages from the new correspondent. This allows removing entries for terminated communications. If Corr2 receives again packets from a moved communication i.e. the redirecting active node has broken down or the route has changed, it forwards the packets towards Corr3. Then Corr3 will execute again the active node discovery procedure.
In the same way, Corr3 will have to execute again the active node discovery procedure in the following case: after a long silence of Corr1, the active node deletes

the entry of the communication in its table; if Corr1 restarts to emit, its packets are received by Corr2 which redirects them towards Corr3.

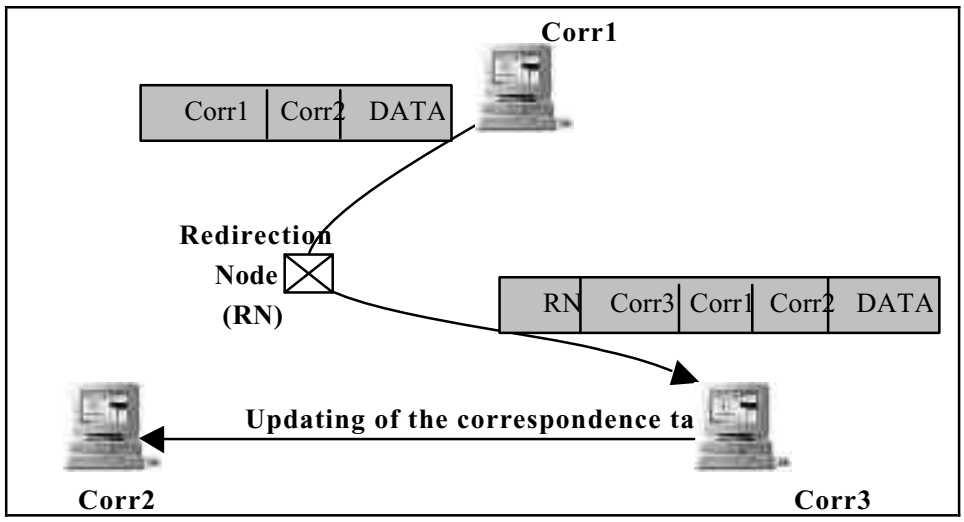


Fig. 5. Redirection of the communication's packets

6.3. Packets Sent from Corr3

Corr3 will need sometimes to send messages to Corr1 because a communication is usually bi-directional. To do this, Corr3 will use a mechanism introduced by the IPv6 mobility: "the home address destination option".

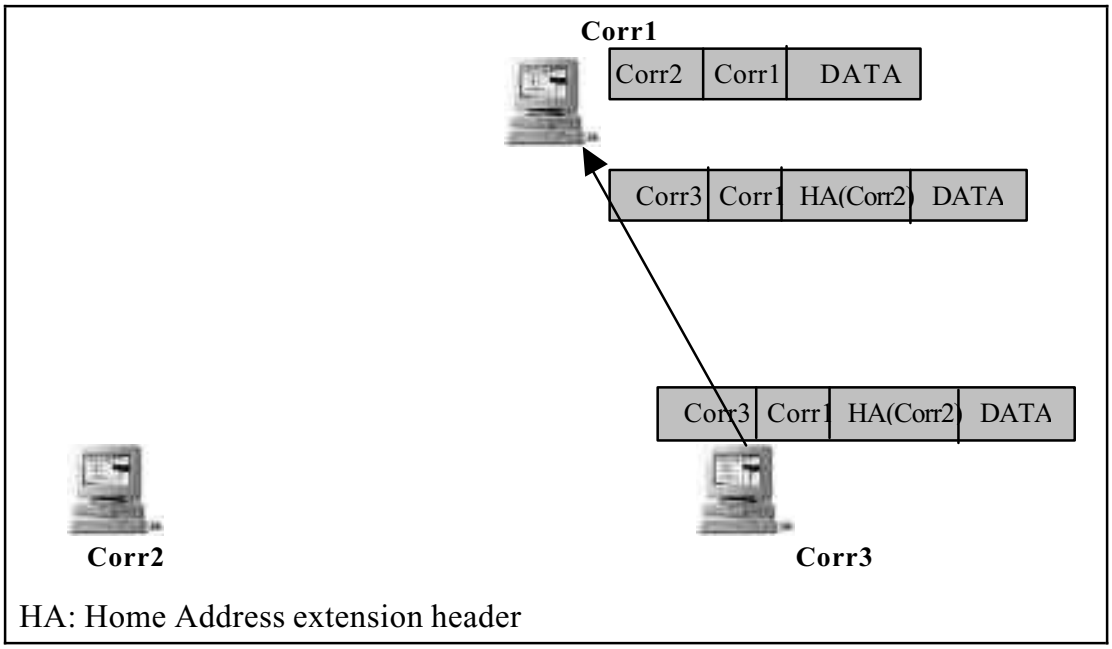


Fig. 6. Packets sent from Corr3

Packets sent from Corr3 to Corr1 in the redirected communication will have Corr3's address as source address and Corr2's address will be placed in the home address destination option. Corr1 will process this packet as if it were arriving from a mobile node.
We notice that the IPv6 mobility support is not mandatory for all the IPv6 nodes. However, the support of the home address destination option is an obligation for all IPv6 nodes. Then, its use is particularly adequate for the realisation of our mechanism.

7. The Initiator of the Communication Displacement and Security

In the communication displacement, like we have said above, Corr2, Corr3 or a third host not contributing in the communication can represent the initiator. Then we have to examine these cases:

- Corr2 is the initiator (terminal currently receiving the communication to displace): Corr2 determines the features of its own communications via it's HA (Host-Agent) and checks if Corr3 accepts to receives the communication by sending it a *comm-recept* request message.

- Corr3 is the initiator (terminal demanding the communication): this case requires the acceptation of Corr2 to give the communication up and that Corr3 is authorized to displace it. This is why authentication and authorization mechanisms must be used in the messages.

- The initiator is represented by a host not contributing in the communication to displace: the user chooses the communication to displace after having the information on the communications of Corr2. But, before proceeding to the displacement, it's needed that Corr2 accepts to give the communication up, and that Corr3 accepts to receive it and is authorized to displace it too which explains the necessity to use authentication and authorization mechanisms in the messages.

These authentication and authorization mechanisms are provided for in the IPv6 protocol.

8. Conclusion and Future Work

In this paper, we presented the concept of communication movement. This notion of redirection of communication is interesting while it can be done transparently for the correspondent Corr1. It doesn't require the interruption of the communication and to establish another one like it already exists in telecommunication networks.

One of the advantages of the mechanisms used is their implementation facility because they are essentially based on the extension header mechanism of IPv6.

In the future, we have to define the mechanism of conversion code load and to extend our work to multicast group communications. Note that the displacement of a multicast communication is not so explicit as displacement of a unicast communication since in the first case it consists for Corr2 to leave the group and to for Corr3 to join it.

We plan also to study the TCP communication redirection and the modifications needed at the transport layer. Indeed, with a TCP communication, the Connection State must be moved contrary to video or voice over UDP flows for example.

References

1. E. Amir, W.Mc Canne and H. Zhang. « An application level video gateway » ACM Multimedia '95, 1995.
2. Markus Breugst and Thomas Magedanz. « Mobile Agents – Enabling Technology for Active Intelligent Network Implementation » IEEE Network Magazine, May/June 1998 Vol. 12 No. 3.
3. Markus Breugst, Lars Hagen, and Thomas Magedanz. « Impacts of Mobile Agent Technology on Mobile Communications System Evolution » IKV++ GmbH, Technical University of Berlin. IEEE Personal Communication Magazine – August 1998
4. Andrew T. Campbell, Herman G. De Meer, Michael E. Kounavis, Kazuho Miki, John B. Vicente et Daniel Villela. « A Survey of Programmable Networks » ACM Computer Communications Review, April 1999.
5. Gisèle Cizault. «IPv6 Théorie et pratique » Editions O'REILLY, Nouvelle édition, Avril 1999.
6. S. Deering and R. Hinden. « Internet Protocol », Version 6 (IPv6) Specification IETF Mobile IP Working Group, RFC 2460, December 1998.
7. Erik Guttman, Charles Perkins, John Veizades and Michael Day. « Service Location Protocol, version 2 » IETF, draft-ietf-srvloc-protocol-v2-12.txt, February 1999.
8. David B. Johnson and Charles Perkins. «Mobility Support in IPv6 » IETF Mobile IP Working Group, draft-ietf-mobileip-ipv6-12.txt, April 2000.
9. Erik Guttman et John Veizades. « Service Location Protocol Modifications for IPv6 » IETF, draft-ietf-srvloc-ipv6-05.txt, October 1998.
10. James Kempf et Erik Guttman. « An API for Service Location » Service Location Working Group, draft-ietf-srvloc-api-08.txt, February 1999.
11. B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, and C. Partridge. « Smart Packets for Active Networks » OpenArch, March 1999 http://www.ir.bbn.com/projects/spkts/smtpkts-index.html.
12. Ethan Solomita, James Kempf and Dan Duchamp. « Xmove, a pseudoserver for X Window Movement » ftp://ftp.cs.columbia.edu/pub/xmove/xmove.thesis.ps, 1995.
13. John Veizades, Erik Guttman, Charles Perkins et S. Kaplan. « Service Location Protocol » Network Working Group, RFC 2165, June 1997.
14. David Wetherrall, Ulana Legedza and John Guttag. «Introducing New Internet Services: Why and How » IEEE Network Magazine, July / August 1998.

An Agent-Inspired Active Network Resource Trading Model Applied to Congestion Control

Lidia Yamamoto, Guy Leduc

Research Unit in Networking, Institut Montefiore, B28, B-4000 Liège, Belgium
yamamoto@run.montefiore.ulg.ac.be
WWW home page: http://www-run.montefiore.ulg.ac.be

Abstract. In order to accommodate fluctuations in network conditions, adaptive applications need to obtain information about resource availability. Using active networks, new models for adaptive applications can be envisaged, which can benefit from the possibility to send mobile code to the network nodes. We describe a model for trading resources inside an active network node, based on the interaction between capsules as reactive user agents, and resource manager agents which reside in the network nodes. We apply the model to the case of a many-to-one audio application with congestion control, which trades off link resources against memory when there is congestion at the outgoing interface towards the destination. Our simulation results indicate that the application makes effective use of the available resources, and it also allows resources to be shared according to user preferences.

1 Introduction

Adaptive applications can tolerate fluctuations in resource availability, and are becoming increasingly important in a strongly decentralised and heterogeneous environment such as the Internet today. In such a complex network, different network technologies and user terminals are interconnected together, and a multitude of applications and services must coexist.

An adaptive application must be able to make optimal use of the available resources, and be able to adapt itself to fluctuations in resource availability. Code mobility as provided by active networks can become a useful tool to help in the adaptation process, since it becomes possible to inject customised computations at optimal points in the network.

A considerable amount of research has been dedicated to algorithms inspired on optimisation and economy theories to control resource usage in networks. In the context of adaptive applications assisted by active networks, the benefit of such techniques is two-fold: on one side, optimal resource sharing configurations can be achieved in a decentralised way; on the other side, it becomes easier to quantify heterogeneity in terms of resource availability, to offer the users the opportunity to trade one type of resource for another.

However, relatively few results have been shown which directly apply such artificial economy models to the specifics of active networks, with special attention to highly adaptive applications. We address this issue in this article by

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 151-169, 2000.

providing a simple model which allows the active applications to make decisions about the amount of resources to use, according to the network conditions found in the active nodes. Using such a model, an audio mixer is developed as an instance of adaptive active network application, which is able to trade bandwidth for memory according to the available prices of each resource.

2 Background

2.1 Active Networks

Active networks (AN) allow the network managers or users to program the network nodes according to their needs, offering a great amount of flexibility. The nodes of an active network [21] are capable not only of forwarding packets as usual but also of loading and executing mobile code. The code can be transported within specialised signalling channels (programmable networks) or within special packets called "capsules" (active networks). Capsules might contain the code itself (such as in [10]) or a reference to it, such that it can be downloaded when the first capsule containing the reference arrives at a given node (such as in [23]). If the distinction between active and programmable networks seemed at some point in time clear [6], the tendency today seems to be towards an integration of the two concepts, since both are forms of achieving open programmability in networks [5], and special flavours in between or combining both approaches are also possible [11].

A framework for an active node architecture is being proposed in [3]. It includes a supporting operating system (the NodeOS), one or more execution environments (EE), and the active applications (AA). The NodeOS is responsible for managing local resources such as CPU processing time, link bandwidth and memory storage. On top of the NodeOS, a number of EEs can be installed. On top of each EE, various AAs can be dynamically loaded and executed. The EE is responsible for controlling the access from the AAs to local resources, and limiting resource usage depending on specified policies.

The NodeOS plays a crucial role in providing access to local node resources, as well as information about resource availability. A NodeOS API is currently being defined [18]. At the moment this API treats four types of resources: computation, memory, communication, and persistent storage. The communication resource is handled through the channel abstraction, which when ready should include QoS support, as well as access to link information such as bandwidth, queue length, and other properties and statistics.

When not all the network nodes are active, it is necessary to discover resources outside an active node. For this purpose, complementary efforts such as CMU Remos [15] could be used. The CMU Remos interface enables network-aware applications to obtain network properties such as topology, latency and bandwidth. Another interesting approach appears in [20], where an *equivalent link* abstraction is proposed, such that from an AN point of view it is possible to look at a set of non active nodes as a single link, with some mechanisms needed to discover the (possibly changing) properties of such a virtual link.

2.2 Mobile Agents and Active Networks

Mobile agents are autonomous pieces of mobile code that travel through the network acting on behalf of their owners. The intersection between mobile agent technology and active network technology is the use of mobile code. The capsules of an active network can be seen as subclasses of mobile agents, specialised for network-related operations.

However, in practice many differences subsist: mobile agents concentrate on application-level duties, and can generally accomplish much more complex tasks than what is generally allowed to capsules. The architectures for mobile agent platforms and active network platforms differ in the kind of support for code mobility that is offered. Mobile agent platforms tend to concentrate on application level services and active network platforms are optimised for transport rather than processing of information.

2.3 Resource management

One of the main difficulties encountered in classical adaptation approaches is how to obtain the required information about resource availability, mainly when this information is hidden in a blackbox network and has to be inferred using only some indirect indications that are observed at the end systems. Using active networks, new models for adaptive applications could be envisaged, which can benefit from the possibility to send capsules or agents to certain elements inside the network. These agents can be in charge of collecting information about network conditions, without having to rely on indirect indications or on heavy signalling protocols. Indeed, the idea of sending small pieces of code directly to where the data needs to be treated, instead of exchanging a large amount of data, is one of the main motivations of mobile agent technology, and it can also be applied to mobile code in the case of active networks.

Actually many adaptation mechanisms come from the world of mobile agents. Some examples are: In [2] the problem of budget planning for mobile agents is addressed, such that they can successfully complete their tasks given their limited budget constraints. In [22] an open resource allocation scheme based on market models is applied to the case of memory allocation for mobile code. In [12] an adaptive QoS scheme for MPEG client-server video applications is described. It is based on intelligent agents that reserve network bandwidth and local CPU cycles, and adjust the video stream appropriately. In [24] a market model to allocate QoS is applied to a conferencing tool targeted at casual meetings where sudden variations in bandwidth availability require an adaptive QoS control strategy.

A lot of work has been done in applying optimisation and economy theories to the distributed allocation of resources in networks [7]. The algorithms derived from such studies are very promising since they converge towards a global optimum in a decentralised way. Several authors (see [14] for an overview) have applied such theories to the problem of end-to-end congestion control, i.e. where

bandwidth is the main scarce resource. The results shown are promising since they are generic enough to be adapted to a wide variety of applications.

Active networks can more easily benefit from such algorithms compared to classical networks, since their code can be dynamically deployed. However, the current AN architectures still offer little support for such algorithms to be implemented in a straightforward manner. The main difficulty is with respect to the interfaces to local information concerning resource availability, which need to be exported to the active applications. In this matter, we can learn from the mobile agents field in order to model the interactions between resource manager agents and user agents. Software agent communication paradigms such as Agent Communication Languages can be helpful, but still need to be specialised to the AN context.

A cost model for active networks is proposed in [17], which expresses the trade-off between different types of resources in a quantitative way. However, the recursive approach adopted makes its use more appropriate in the context of reservation-based applications, instead of highly adaptive ones.

3 Resource Trading Model

In previous work [25] we studied the potential and limitations of active networks in the context of adaptive applications, with a case study on a layered multicast application. These first results led us to propose a protocol that we called Active Layered Multicast Adaptation (ALMA) [26]. As a side effect of designing ALMA, a slightly different model of an active node popped up, in which the intelligence is distributed among the active applications representing the user interests, and the active resource managers representing the network provider interests. Market-based techniques are naturally suitable for such a model. In the present paper we push the idea forward by analysing the impact of its various facets on the domain of adaptive applications over active networks.

Our model aims at offering a generic communication abstraction between active network agents, such that different adaptive applications and different resource management policies can be implemented. We are mainly concerned with adaptation to available resources when resources cannot be reserved in advance, either because the router itself does not support reservations, or because the network is heterogeneous and some of the routers along the path offer no QoS support. The idea is to enable auto-configurable applications and resource managers, such that the code from both types of agents can be dynamically loaded, in order to make them evolve to adapt to new conditions.

Figure 1 shows the model as it could be implemented over the DARPA AN architecture.

In the model, two types of agents communicate to seek an equilibrium. Each agent tries to optimise its own benefits: on one side resource manager agents have the goal of maximising resource usage while maintaining a good performance level. On the other side, user agents try to obtain a better quality/price relation for the resources consumed, and to efficiently manage their own budgets avoiding

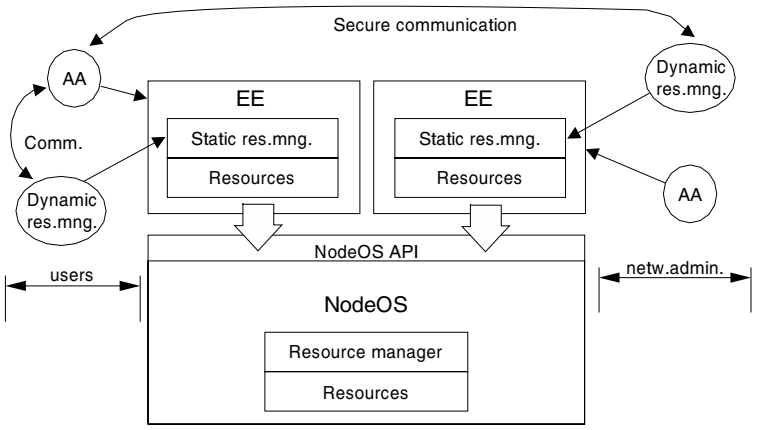


Fig. 1. Resource managers and active applications. Hypothetic placement over the DARPA AN architecture

waste. Both types of agents are implemented as AAs with different privileges, and they communicate such that the resource managers can "sell" resources to the user agents at a price that varies as a function of the demand for the resource.

A currency is introduced into the system to allow for trading of different resource types. This is the basic requirement for artificial economy models such as [2, 7, 22] to develop in active nodes, and is also an essential feedback parameter for most algorithms based on optimisation (e.g. [14]).

Such a model is per se not entirely new, and is in fact a mere simplification of existing artificial economy models which have been mainly applied to the agents domain. For example, in [9] a similar although more complex model is presented, and applied to a circuit switching set-up of paths (reservation-based). The main difference with respect to our model is that we try to adapt it to the specifics of adaptive applications over active networks (no resource reservations), in which reaction time is critical, therefore precluding the use of complex transactions.

3.1 Resource Manager Agents

Resource managers export resource prices which are a function of the resource utilisation. The utilisation is related to the load, and to the demand for a resource. The function or algorithm used to calculate prices can be shaped to implement desired policies, such as to achieve high utilisation, but also to offer good quality to the users. Resource managers may contain dynamic and/or static code. Some lower level functions which are especially time-critical might be implemented using static code (or even in hardware) while the mobile part would be used to implement more complex policies and to select from a set of pre-existent lower level functions. It is necessary to have the possibility to use dynamic code in order to be able to improve strategies, that is, make them evolve over time, in an active network. Here is one of the places where the alliance between AN and mobile agents can become a must: resource managers

can be deployed using mobile agents that are sent by the network manager in order to install new policies. Classical mobile agents for network management can be used for this purpose.

Resource managers are implemented as AAs injected by the network provider, which are executed with network administrator's privileges. There is one class of resource manager for each type of resource concerned. The most relevant classes are: link managers, CPU managers, and memory managers. We will give some more attention to link managers, since they play a crucial role in network congestion control.

A link manager is a resource manager responsible for the resources of a link, such as bandwidth and outgoing queue space. It exports a link price which is a function of the current load on the link. The price works as a congestion indication [14]. Since the link managers are controlled by the network provider, as the other resource managers, the shape of the function is a provider's choice, and can be updated to achieve a desired effect, such as to encourage or discourage bursty traffic, to achieve a certain level of utilisation, to converge faster or smoother, etc. For example, a binary feedback scheme such as the one provided by RED active queue management [8] can be regarded as a special case of pricing indication for TCP/IP networks. A RED gateway avoids bias against bursty traffic by dropping or marking packets with a probability that is a function of the average queue size. In [13] a marking scheme called REM is proposed, which is based on optimisation results and is shown to achieve better throughput and fairness than RED, while at the same time presenting no packet losses. In [1] it is shown that faster convergence to the optimum bandwidth shares can be achieved by only changing the price adjustment algorithm from the simple gradient projection used in [14] to a scaled gradient projection using an approximate Newton method.

Link manager agents also allow abstractions to be made which enable the user applications to adapt to a wide variety of environments in a transparent way. By exporting prices instead of the link internal state information directly, it is possible to hide the specific details of link characteristics while at the same time offering a proper congestion indication to adaptive applications. For example, the price function for a classical point-to-point link would be different from that of a wireless link, where the local interface load is not a good indication of the actual link utilisation.

An important feature of the link manager price abstraction is that it allows the active applications to deal with non-active nodes in a transparent way. For example, a link manager that implements the equivalent link abstraction [20] could export a price which would be a function of the estimated average rate, average delay and packet loss probability, which are changing properties in the case of an Internet virtual link. This type of abstraction is crucial for the success of active networks, since the deployment of active nodes will depend on their ability to complement and interwork with existing technologies.

A lot of research is still to be done on how to adjust prices in artificial agent-based economies. An interesting analysis can be found in [16]. For this

paper however this will not be our focus. We will rather concentrate on the user side, assuming that the resource manager agents in the active nodes are able to implement suitable pricing algorithms.

3.2 Capsules as Reactive User Agents

The second type of agents in the model are the active packets themselves, modelled as simple reactive mobile agents. Actually capsules can be regarded as a specialised subclass of mobile agents. They travel to network nodes where they decide when to continue or stop the trip (e.g. stop due to congestion), when to fork new capsules (e.g. in a multicast branch), and which amount of resources to use at each node in order to complete their (generally simplified) tasks.

Capsules have the properties of autonomy and mobility, but need to be simple enough to be executed at the network layer, where performance is often critical. They need to take fast decisions using little data and reasoning, therefore they must be extremely reactive. They can therefore be classified in the reactive agents category. This does not exclude from the model the possibility of using more intelligent mobile agents, jointly with capsules. However, in this paper we focus only on capsules.

Capsules generally represent the user interests, with the goal of attaining the best possible quality at the lowest possible costs. This means that a capsule must be able to make rational decisions on how to spend its limited budget, after consulting resource manager agents for information about prices of the various resources need.

Capsules carry a budget that allows them to afford resources in the active nodes, as the resource limit field in ANTS [23]. It looks like a TTL (Time-To-Live) field which is decremented by n units for the consumption of a certain amount of resources. When it reaches zero the capsule is discarded. As explained in [23], the budget field must be protected such that the capsules themselves cannot modify it. Again, since capsules are reactive agents, the transactions must be kept simple enough. Instead of the auction mechanisms frequently used by agents [7], the capsules will most of the time either accept or refuse to pay a given price. Refusing might imply that the capsule simply disappears from the system, because it does not have enough budget to proceed its journey to its destination. Again, we do not preclude the usage of more complex mechanisms implemented by intelligent mobile agents. These agents can also have access to the resource manager interfaces, therefore they can also benefit from the model, however they are not our focus as pointed out earlier in this section.

At the end systems (hosts) conventional programs or intelligent agents can be used to spawn capsules. Since these hosts can dedicate significant amount of resources to information processing, sophisticated strategies can be used to decide which capsules to spawn and when, and how much of the total user's income can be assigned to each capsule (planning). These tasks are then not delegated to the capsules themselves, but can be delegated to more generic mobile agents such as the ones described in [2].

Here we face the problem of how the budget should be distributed between capsules for one user, and among different users, and also of how to make purchase decisions according to budget constraints. This can be done with the help of utility functions, which quantify the level of user satisfaction for receiving a certain amount of a good, or more generally a combination of goods (market basket) [19].

According to economics, a typical behaviour of a rational consumer agent is to try to maximise its utility subject to the budget constraints given by their limited income.

For an example where only two goods are involved, the user optimisation problem is then typically expressed as:

$$Maximise\ U(x,y) \tag{1}$$

subject to:

$$p_x x + p_y y = I \tag{2}$$

where $U(x,y)$ is the utility function, x and y are the quantities of two goods in their respective units, p_x is the price per unit of x, p_y the price per unit of y, and I is the user's income.

Such a maximisation process will lead to an equilibrium if the utility function satisfies some properties such as being strictly concave increasing, i.e. the increase in satisfaction is smaller the more items of one good are consumed. The increase in satisfaction that a user obtains from consuming one more item of a given good is called the marginal utility. Within the rational consumer assumption, the marginal utility is a decreasing function of the number of items of a given good. This is called the diminishing marginal utility assumption, and it is directly related to the demand function of a given user for a given good. It means that the more one has from something, the less it is willing to pay to obtain more of it.

A typical utility function is the Cobb-Douglas utility function, given by:

$$U(x,y) = a \cdot log(x) + (1-a)log(y) \tag{3}$$

where $0 \leq\ a \leq 1$ is a constant which represents the importance the user assigns to x with respect to y.

Applying the method of Lagrange multipliers, the solution of the user optimisation problem with the Cobb-Douglas utility function is given by [19]:

$$x(p_x) = \frac{a \cdot I}{p_x} \tag{4}$$

$$y(p_x) = \frac{(1-a)I}{p_y} \tag{5}$$

The functions $x(p_x)$ and $y(p_x)$ are the demand functions for goods x and y given their current prices per unit p_x and p_y respectively. With demand functions shaped like these, it is possible to chose the quantity of a given resource to

consume in order to maximise user satisfaction, given the current price for the resource and the agent's income. Since the price information for each type of resource is available in the active nodes, it is possible for an agent to calculate the amount of resources it can consume at each node, given only an upper bound on the budget per hop it has planned to spend. In the case of capsules, the planning decisions to calculate this upper bound could be performed either at the hosts or by more intelligent mobile agents that would be sent to the active nodes less frequently than capsules. The techniques for such planning are out of scope of this article, and the interested reader can refer to [2] for more information.

The same reasoning can be easily generalised to an arbitrary number of resources [7]. However, for the purpose of our study this will be sufficient, since we will restrict ourselves to link and memory resources.

Another important characteristic of the Cobb-Douglas function is that it allows us to easily quantify the preferences of different consumers towards one good. For example, given two user agents u_1 and u_2 with the same income I, if agent u_1 has the weight a_1 on its utility function for good x, and agent u_2 has weight $a_2 = n \cdot a_1$ for the same good, we have:

$$x_1(p_x) = \frac{a_1 \cdot I}{p_x} \tag{6}$$

$$x_2(p_x) = \frac{a_2 \cdot I}{p_x} = \frac{n \cdot a_1 \cdot I}{p_x} = n x_1(p_x) \tag{7}$$

Thus for a given price p_x we have:

$$a_2 = n \cdot a_1 \Rightarrow x_2 = n \cdot x_1 \tag{8}$$

It means that if we know that agent u_2 values resource x twice as much as agent u_1, then when faced with the same price, u_2 will get twice as much of x as u_1. This is an interesting tool for computer network applications, since it provides a quantitative way to provide service differentiation according to the preferences of users. This capability is already known from literature, e.g. [7, 14]. For example, when trading bandwidth for memory storage, a time-constrained application such as an audio-conference will certainly prefer bandwidth to storage if prices are the same, while a bulk transfer application would probably prefer to store as much information as possible when the links are congested, in order to avoid losses and retransmissions.

In real world economies it is difficult to quantify utilities, but in artificial economies this might be less difficult. Without having to relate artificial currencies to real ones, we could imagine that the maximum budget per unit of time is controlled by a policy server from the network provider that the user is subscribed to, in order to guarantee that users will employ such budget rationally and prevent malicious users from grabbing most of the resources by marking all their capsules with a high budget. This can be compared to the IETF diffserv policies. However, in diffserv only a few predefined classes of service are available, while with such artificial economies, a whole range of classes could appear (and eventually die out), defined by their particular utility characteristics.

4 Congestion Control for a Concast Audio Mixer

We illustrate the use of the trading model through a congestion control scheme for a many-to-one (concast) service. The concast example shows capsules that trade bandwidth for memory when there is congestion.

The term "concast" has been defined in [4] as a many-to-one service, in opposition to multicast (one-to-many). Figure 2 illustrates this concept. While multicast copies information from one source to many destinations, concast merges information from several sources to one destination. A concast service can be used, for instance, to aggregate feedback in a reliable multicast service, to transmit reception statistics in a multimedia session, to merge information coming from several sources in an auction or tele-voting application, or to combine several real-time streams into one, e.g. to perform audio mixing from several audio sources.

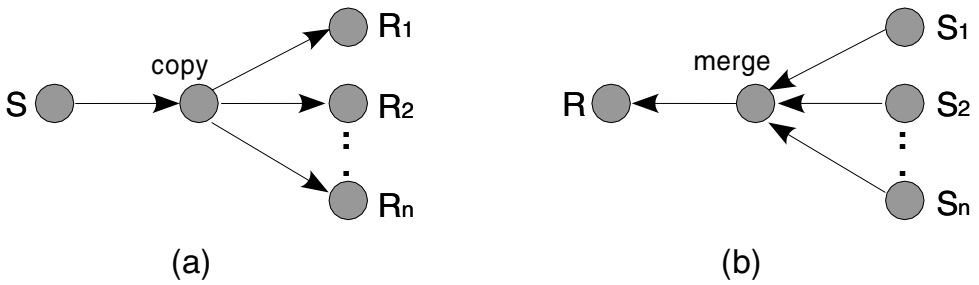


Fig. 2. Multicast (a) versus Concast (b) service abstractions.

The concast service is faced with the feedback implosion problem, that is, multiple simultaneous sources might congest the path to the single destination, if no congestion control is performed. This problem is aggravated by the fact that many concast flows are used as signalling to support a more robust protocol such as the aggregation of NACK feedback messages for a reliable multicast protocol. Flow control for such signalling messages is often neglected or oversimplified, since the signalling traffic is assumed to be kept small enough when compared to the data traffic. This might lead to poor performance when there is congestion in such a signalling path. We argue that for future sophisticated active services congestion control will be equally important on the signalling and data paths, since the distinction between both tends to become naturally blurred as we approach new network service composition frameworks which are not simply stack-based as the classical OSI model.

For the purpose of this study, we focus on the case of an audio mixing application. However, the same ideas are applicable to any application making use of the concast service, differing only in the way that packets are combined (merge semantics).

In our audio mixing application, several sources generate audio streams in a session. The streams are collected at a single node, which can either record them, play them back or redistribute them to a multicast group. If the receiving node collects all the data and mixes it locally, it might end up with the implosion problem. It is possible to perform mixing operations at every active node, so that the receiver gets a single stream already mixed. However this might cost too much processing and/or memory space in the active routers, so it might become too expensive. Also note that mixing streams delays them, since it is necessary to wait until a minimum number of packets arrive in order to sum up their audio payloads.

The mixing operation is a physical sum of audio samples. Therefore the resulting audio payload has the same number of bytes as each of the original payloads. We assume that there is a maximum amount of signals that can be added up without saturation. Another assumption is that a constant bit rate codec is used, with no silence suppression, such that the signals are generated at a constant rate from the beginning to the end of the session. This initial rate will be altered along the path, as capsules are mixed according to the network conditions.

Note that the mixing operation requires a list of addresses, in order to identify the list of sources already mixed. One might argue that such a list might also occupy bandwidth, since the packet size has to increase in order to accommodate it. But if an estimation of the group size is available, a fixed-size bitstream can be used to hold the list of sources. Adding or removing a source becomes as simple as setting or resetting a bit in the bitstream. The intersection and union operations can also be implemented as AND and OR binary operations respectively.

We propose to trade bandwidth for processing and memory space to achieve a compromise solution which uses resources efficiently and therefore is able to control congestion. The idea is that when there is congestion at an outgoing interface, the consequently high bandwidth prices will push the users to save bandwidth by performing mixing of data. When the congestion clears up, the users can again benefit from the available bandwidth to avoid the extra delays imposed by the mixing operation.

This application needs only one type of capsule: the audio data capsule, which carries the audio payload from a source, or a list of sources, to the destination. The capsule consults the link managers and memory managers in the active nodes along the path, in order to decide whether to mix with other capsules of the same session, or to proceed its journey to the destination. For simplification purposes, the CPU manager is not involved. We assume that CPU costs are either negligible or can be combined with memory costs.

When a capsule arrives at a node, the first thing it does is to check whether another payload coming from the same source is already buffered. In this case, the arriving capsule cannot mix its payload to the buffered one, which carries earlier samples. Note that this check is in fact an intersection operation to check whether one of the sources mixed in the current capsule is already buffered. If that is the case, the arriving capsule immediately dispatches the buffered payload by

creating a new data capsule and injecting it into the local execution environment. This procedure also prevents misordering of packets. As a consequence, at most one packet payload per session is buffered in an active node. We assume that all payloads have the same size.

After that, the capsule can take a decision to either proceed, buffer its payload (for mixing), or discard itself. This decision depends on the budget it carries, and on the prices of the memory and link resources it needs. A number of alternative decision strategies can be envisaged:

Null strategy: Corresponds to the trivial case when no congestion control is performed. In this case the capsule always goes intact to the outgoing interface.

First strategy: If there are other audio payloads from the same group waiting in the memory buffer, it adds its own payload to the buffered one and adds its list of sources to the list of sources already buffered (union operation). Then it terminates execution. If no other audio payloads from the same group are buffered yet, it decides for the cheapest resource: if the price of memory (to buffer the payload for future mixing) is currently lower than the transmission price for the capsule, then it decides to buffer itself; otherwise it decides to move on to the next hop.

Second strategy: The arriving capsule mixes its own data with the buffered one (if existent), then it chooses the cheapest resource, either memory or bandwidth.

The difference between the first strategy and the second strategy is that the first strategy always decides for storage when another payload from the same session is already buffered, while the second strategy always chooses the cheapest resource, independent on the fact that another payload is already buffered or not.

Although the first and second strategies are very naive, they already give quite reasonable results as we will see below. However, their behaviour is suboptimal and they do not take into account the different preferences for resources.

Third strategy: It first mixes its own data with the buffered one (if existent), as in the second strategy. It obtains a new payload that combines samples from n sources (n is known from the list of source addresses). It also knows N, the maximum number of sources that can be mixed together without saturation. Then it calculates the amounts of link and memory resources according to equations 4 (for link) and 5 (for memory), where the a parameter is also carried in the capsule, and expresses its preferences for link resources with respect to memory. It then tries to keep the resources in the proportions obtained, as follows:

If $\frac{N-n}{n} > \frac{x}{y}$ then store, else move on.

Where x is the demand for link resources according to equation 4, and y is the demand for memory resources according to equation 5.

Since $\frac{x}{y} = \frac{a \cdot p_y}{(1-a)p_x}$, the decision is independent on the capsule's budget I. The a parameter will play a role in the proportion of link resources used with respect to memory resources. The higher a, the higher the amount of link resources used,

and therefore less capsules will be mixed together, for given memory and link prices.

Note that in all cases, capsules that run out of budget are automatically discarded by the resource managers, thus there is no need to explicitely indicate this operation.

5 Simulations

The audio mixing AA has been simulated with the help of an AN module that we developed for the NS simulator [27]. This module implements a simplified AN architecture consisting of a NodeOS, an EE, and some resource managers. The simulated EE executes capsules written in TCL language.

The topology for the simulations is shown in figure 3. It consists of n sessions of m sources and one receiver each. The sessions traverse a bottleneck (link L), so that the capsules in active node N must decide to mix or to proceed intact to the receiver node, according to the prices of link or memory resources available from the resource managers.

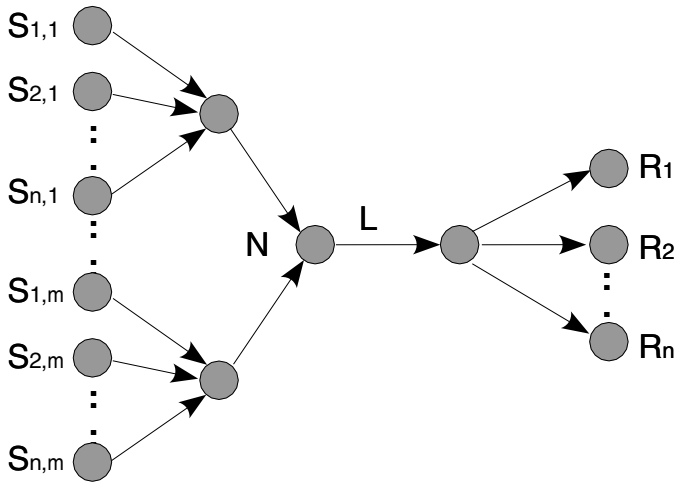


Fig. 3. Topology used in the simulations.

The price function used is based on the one in [22]:

$$price = 1000 \cdot \sqrt{\frac{1.01}{(1.01 - load)}}) - 1000 \tag{9}$$

This function is a practical implementation of a convex increasing function that would go to infinite as the load approaches 100%. It forces the price to rise sharply as we approach high loads, which discourages applications from using a

resource when its load is too high. This gives the applications a clear indication of the "dangerous zone" to avoid, while at the same time encouraging a relatively high utilisation.

In the case of the memory manager, the load is given by the ratio between the average number of memory units occupied, and the total number of memory units available to user capsules (which is of course assumed to be much smaller than the actual amount of memory available). The average is calculated using an exponential weighted moving average (EWMA [8]).

For the link manager, the load is given by the average queue occupancy ratio at the outgoing link interface. This average ratio is obtaining by calculating the average queue length as an EWMA, and then dividing by the maximum queue size. The resulting congestion indication is a bit similar to RED, except that here the binary feedback is replaced by an explicit price indication to the arriving capsules. Note that the actual usage of bandwidth is not taken into account in the price function, if it does not cause queues to build up. This is therefore a very simplified version of a link manager, but it already serves the purpose of controlling congestion.

We first run an example where 2 sessions are active. There are 5 sources per session, each sending an audio stream of 100kbps to a single receiver, resulting in a total of 1Mbps of traffic arriving at N. The capacity of link L is set to 500kbps.

Figure 4 (top) shows what happens in the trivial case when no congestion control is used. In this case, link overflow occurs at L, and half of the packets are dropped. The remaining packets receive an unequal share of the bottleneck link as we can see in the figure.

Figure 4 (middle) shows what happens when the first strategy is used. First we can notice that the two sessions (left and right) get approximately the same share of the bottleneck. Additionally, no packet losses were observed during the simulation. However, the link is underutilised. This can be explained by the fact that this strategy always favours memory when there is already an item in memory.

The second strategy is a bit more clever (Figure 4, bottom). It always chooses the cheapest resource, either memory or bandwidth. Therefore it is able to grab any bandwidth when it becomes available. Here again, no packet losses occur. However, with this strategy it is not possible to specify different weights for each resource.

Now let us look at the third strategy, which allows us to specify utility weights through the a parameter. Figure 5 (top) shows its behaviour when the weights of the two sessions are the same. We see that this strategy is able to share the bandwidth efficiently. It also leads to more stable rates when compared to the previous strategies. The same happens when the weights are different (bottom side of Figure 5), but in this case each session receives link resources in proportion to its respective weight, expressed by the a parameter value.

Until this point only the rates have been shown, since our main goal is to achieve congestion control. Table 1 shows average memory and link parameters taken over the complete duration of the simulations shown in figures 4 and 5. The

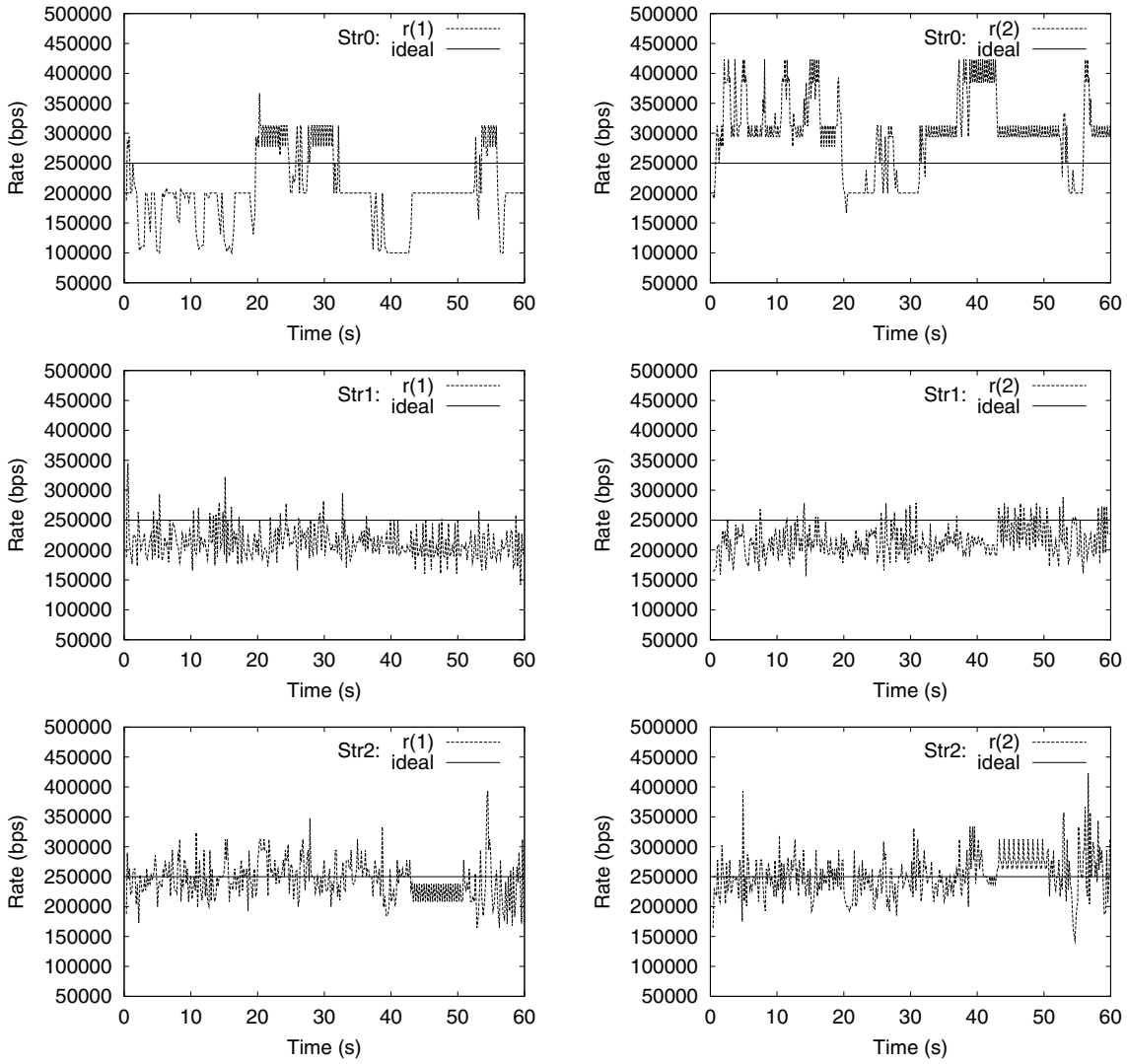


Fig. 4. Evolution of rates in time for 2 sessions, as perceived by their respective receivers. Left: first session (receiver r_1). Right: second session (receiver r_2). Top: no congestion control. Middle: first strategy. Bottom: second strategy

null strategy (Str.0, no congestion control) occupies most of the link resources and causes the link price to rise, since the link queue is most of the time full. Strategy 1 reduces link utilisation by using memory for mixing, however it does that in an inefficient way when compared to strategy 2, which is able to use more bandwidth while decreasing both link buffer occupancy and memory utilisation. The result is a decrease in link and memory prices.

Strategy 3, when $a_1 = a_2$ (Str.3a in Table 1), improves further by keeping the average link queue occupancy at a very low level, in spite of a high bandwidth utilisation. This can be explained by the fact that this strategy tries to find an optimum balance between the usage of memory (which delays packets) and the usage of link resources. It therefore waits for the good moment to send a packet over the link, the moment when link prices are favourable (which in our case corresponds to low queue occupancy).

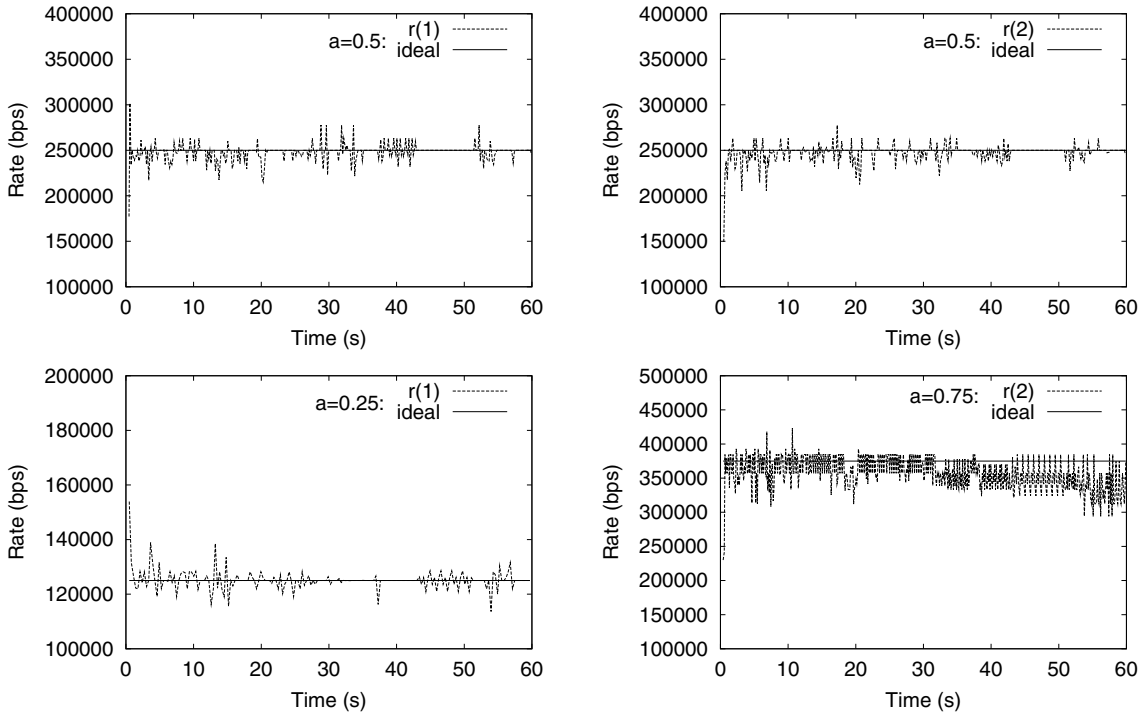


Fig. 5. Evolution of rates in time for 2 sessions using the third strategy. Top: $a_1 = a_2 = 0.5$ for link resources. Bottom: $a_1 = 0.25$, $a_2 = 0.75$

Table 1. Memory and link usage parameters for different strategies

Parameter (average)	Str.0	Str.1	Str.2	Str.3a	Str.3b
memory utilisation (%)	0	7.18	5.14	4.98	5.33
bandwidth utilisation (%)	99.91	84.54	98.18	98.93	96.37
link queue occupancy (%)	93.58	5.83	4.97	1.61	2.98
memory price ($)	0	37.71	26.57	25.62	27.37
link price ($)	2914.60	32.35	27.71	10.17	17.22

Column Str.3b in Table 1 shows the average resource parameters when $a_1 \neq a_2$, corresponding to the situation illustrated at the bottom side of Figure 5. There are no significant changes in the parameters at node N, with respect to Str.3a, but the prices are higher, which is an unexpected result, as we would expect the two sessions to concentrate each one on its respective preferred resource, making the load equally distributed. We are still investigating the reason for this discrepancy.

In order to have a better insight on the mixing procedure, we now look at the number of audio sources carried by each capsule that arrives at its destination. Table 2 shows the percentage of packets that arrived at both receivers, over the total number of packets sent, which is the mix of a given number of sources. The first row ("0 (lost)") represents lost packets. The second row (one source) represents the percentage of packets that arrive intact from the source. The third row (two sources) represents packets that mix samples from two different sources,

Table 2. Average percentage of packets carrying the sum of samples from a given number of sources

# sources	Str.0	Str.1	Str.2	Str.3a	Str.3b(r1)	Str.3b(r2)
0 (lost)	49.61	0	0	0	0	0
1	50.39	66.22	39.87	1.54	0.05	60.86
2	0	0.02	33.53	96.16	0.48	39.14
3	0	0	15.80	2.30	1.93	0
4	0	0.43	6.26	0	97.54	0
5	0	33.34	4.53	0	0	0

and so on for the rest of the rows. The columns represent the strategies used. For the null strategy, roughly half of the packets are lost, and the remaining packets arrive intact (no mixing). Strategy 1 is an all-or-nothing strategy: two thirds of the arriving packets contain data from only one source, while one third contains data from all the sources for a given session. The second strategy distributes the mixing effort more evenly.

As for the third strategy, column Str.3a of Table 2 shows the results when $a_1 = a_2$, corresponding to the simulation result shown at the top side of Figure 5. Columns Str.3b(r1) and Str.3b(r2) show the results for receivers r_1 and r_2 respectively, when $a_1 \neq a_2$ (Figure 5, bottom). We can see that when $a_1 = a_2$, most of the packets arrive at the receivers containing samples from two sources mixed together, while for $a_1 = 0.25$ (r_1), most of the packets contain samples from four sources, and for $a_2 = 0.75$ (r_2), more than one third of the arriving packets contain a mix of only two sources, and the rest only one source (no mixing). This shows that strategy 3 tries to stabilise at a target mixing level, which is characterised by the a parameter. The lowest the a parameter value is for a given session, the highest the mixing level which is achieved.

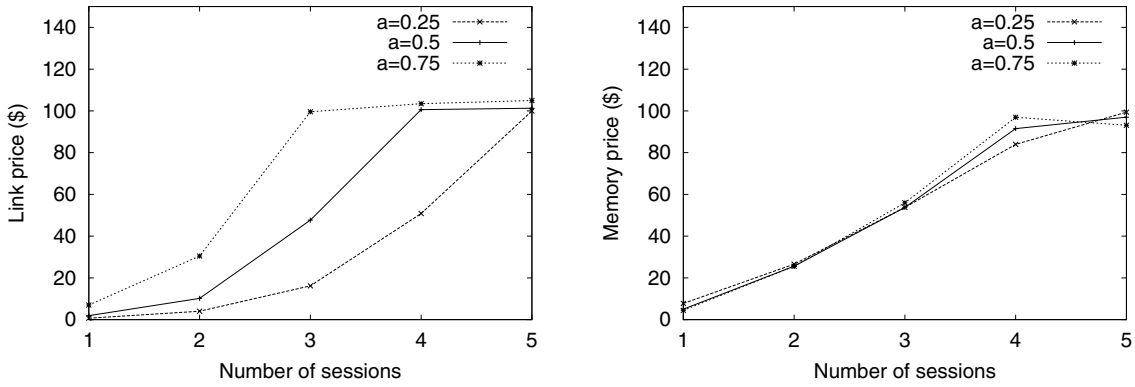


Fig. 6. Average link and memory prices when varying the number of sessions in parallel.

Finally, we vary the number of sessions in parallel using the third strategy, in three separate runs: during the first run all sessions have $a = 0.25$, during the

second, $a = 0.5$, and the third, $a = 0.75$. The total average prices for memory and link buffer occupancy are depicted in Figure 6. We can see that the a parameter has a clear influence on the link prices, that increase with a for a given number of sessions, as expected. However it has little influence on the memory prices. This can probably be explained by the fact that, although the application avoids using the memory during a too long period due to delay constraints, at any time each session has at most one packet stored in memory. In the simulations shown, memory does not become a bottleneck, therefore the small impact on prices.

6 Conclusions and Future Work

We have described a model for trading resources inside an active network node, which draws many elements from agent technology. We apply it to a concast audio mixing application which trades off link resources against memory in the presence of bottleneck links. The concast application is able to take congestion control decisions locally at each active node, such that no closed loop feedback between source and destination is needed. Using simulations, we have studied three different strategies to make a decision on the amount of resources to use: two naive strategies based on the cheapest price, and a strategy that makes use of utility function weights. The results indicate that first two strategies are already able to make improvements over the case when no congestion control is used, but they use resources inefficiently. The third strategy gives better results, achieving a stable and efficient sharing of resources.

We have several research directions to pursue: the most immediate one is to perform more complex simulations involving multiple node and link types, resource manager types, active and non-active nodes, different user strategies, etc. An implementation over a real active networking platform is also envisaged for the near future. We also plan to investigate the issues of dynamic resource manager upgrade with the help of mobile agents. The precise communication abstractions among the various kinds of agents need further attention too.

References

1. S. Athuraliya, S. Low, "Optimization Flow Control with Newton-Like Algorithm", Journal of Telecommunication Systems, to appear, 2000.
2. J. Bredin et al., "A Game-Theoretic Formulation of Multi-Agent Resource Allocation", Proceedings of the 2000 International Conference on Autonomous Agents, Barcelona, Spain, June 2000.
3. K.L. Calvert (ed) et al., "Architectural Framework for Active Networks", (DARPA) AN Working Group, draft version 1.0, July 1999, work in progress.
4. K. Calvert, "Toward an Active Internet", Active and Programmable Networks Miniconference, Networking 2000, Paris, France, May 2000.
5. A.T. Campbell, et al., "A Survey of Programmable Networks", ACM SIGCOMM Computer Communication Review, April 1999, p.7-23.
6. T.M. Chen, A.W. Jackson, "Active and Programmable Networks", Guest Editorial, IEEE Network, May/June 1998, p.10-11.

7. D. F. Ferguson, C. Nikolaou, J. Sairamesh, Y. Yemini, "Economic Models for Allocating Resources in Computer Systems", Market based Control of Distributed Systems, Ed. Scott Clearwater, World Scientific Press, 1996.
8. S. Floyd, V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, August 1993.
9. M.A. Gibney, N.J. Vriend, J.M. Griffths, "Market-Based Call Routing in Telecommunication Networks Using Adaptive Pricing and Real Bidding", LNAI n.1699, Proceedings of the IATA'99 Workshop, Stockholm, Sweden, August 1999.
10. M. Hicks et al., "PLANet: An Active Internetwork", Proceedings of IEEE INFOCOM'99, New York, 1999.
11. G. Hjlmtýsson, "The Pronto Platform: A Flexible Toolkit for Programming Networks using a Commodity Operating System", Proceedings of IEEE OPENARCH'2000, Tel-Aviv, Israel, March 2000, p. 98-107.
12. K. Jun, L. Blni, D. Yau, D.C. Marinescu, "Intelligent QoS Support for an Adaptive Video Service", To appear in the Proceedings of IRMA 2000.
13. D. Lapsley, S. Low, "Random Early Marking for Internet Congestion Control", Proceedings of Globecom'99, pp. 1747-1752, Rio de Janeiro, Brazil, December, 1999.
14. S. Low, D.E. Lapsley, "Optimization Flow Control, I: Basic Algorithm and Convergence", IEEE/ACM Transactions on Networking, 1999.
15. N. Miller, P. Steenkiste, "Collecting Network Status Information for Network-Aware Applications", Proceedings of IEEE INFOCOM'2000, Tel-Aviv, Israel, March 2000.
16. H. Mizuta, K. Steiglitz, E. Lirov, "Effects of Price Signal Choices on Market Stability", 4th Workshop on Economics with Heterogenous Interacting Agents, Genoa, June 4-5, 1999.
17. K. Najafi, A. Leon-Garcia, "A Novel Cost Model for Active Networks", Proc. of Int. Conf. on Communication Technologies, World Computer Congress 2000.
18. L. Peterson (ed) et al., "NodeOS Interface Specification", (DARPA) AN NodeOS Working Group, draft, January 2000, work in progress.
19. R.S. Pindyck, D.L. Rubinfeld, "Microeconomics", Forth Edition, Prentice Hall International Inc., 1998.
20. R. Sivakumar, S. Han, V. Bharghavan "A Scalable Architecture for Active Networks", Proceedings of IEEE OPENARCH'2000, Tel-Aviv, Israel, March 2000.
21. D. L. Tennenhouse et al., "A Survey of Active Network Research", IEEE Communications Magazine, Vol. 35, No. 1, pp80-86. January 1997.
22. C. Tschudin, "Open Resource Allocation for Mobile Code", Proceedings of the Mobile Agent'97 Workshop, Berlin, Germany, April 1997.
23. D. J. Wetherall, J. V. Guttag, D. L. Tennenhouse, "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols", Proceedings of IEEE OPENARCH'98, San Francisco, USA, April 1998.
24. H. Yamaki, M.P. Wellman, T. Ishida, "A market-based approach to allocating QoS for multimedia applications", Proceedings of ICMAS'96, Kyoto, Japan, December 1996.
25. L. Yamamoto, G. Leduc, "Adaptive Applications over Active Networks: Case Study on Layered Multicast", Proceedings of ECUMN'2000, Colmar, France, October 2000.
26. L. Yamamoto, G. Leduc, "An Active Layered Multicast Adaptation Protocol", Proceedings of IWAN'2000, Tokyo, Japan, October 2000.
27. *UCB/LBNL/VINT Network Simulator - ns (version 2), http://www-mash.cs.berkeley.edu/ns/.*

On Synchronization in a Mobile Environment

Anca Rarau, Ioan Salomie, and Kalman Pusztai

Department of Computer Science
Technical University of Cluj-Napoca
3400, Cluj-Napoca, Romania
{Anca.Rarau, Ioan.Salomie, Kalman.Pusztai}@cs.utcluj.ro

Abstract. In a distributed environment, static and mobile behaviors interleave at both logical and physical level. In this paper we identify not only those elements that can feature mobile behaviour but also types of such behaviors at each of the two levels. We believe that the synchronization between the mobility at the logical level and the one at the physical level is a major requirement of a mobile environment. This paper discusses authors' view on synchronization - passive and reactive behavior. A mathematical description of a reactive behavior is presented also. Finally a survey of existing mobile systems from the synchronization point of view is presented.

1 Introduction

A mobile environment is a system in which the components change their positions one to the other. The communication between them does not rely on the same assumption one can make for static environment, namely every component is uniquely identified by its position. As a result, *location* [9] becomes extremely important in a mobile environment. In order to specialize the above definition for distributed systems we have identified the elements that feature mobile behaviors, computers / devices, users and software. Hereinafter, we will call computers / devices using a single word, namely machine. The mobility of the three entities take place at the physical and logical level, but besides mobile behaviors the static ones can be also identified at the two levels. Table 1 depicts all possible static-mobile combinations.

The mobility that appears at logical level is called logical mobility – the migratory entities, code, code & data, code & data & state are referred to as logical entities. On the other side, physical mobility takes place at physical level and includes machines and users as migratory entities.

The rest of the paper is organized as follows. In Section 2 we introduce taxonomy for mobility. In Section 3 we state that synchronization between logical and physical levels is a great challenge for mobile environment and identify the solutions for it. A mathematical description of reactive behavior is presented in Section 4. Section 5 contains a survey concerning how the synchronization is tackled in the existing mobile agent systems. In Section 6 we draw the conclusions and discuss future work.

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 171-183, 2000.

Table 1. Static-mobile combinations

	Physical level	Logical level	
Fully static	Static	Static	Does not assume mobility aspects
Partial mobile	Static	Mobile	Migration at the logical level takes place in a static physical environment
Partial mobile	Mobile	Static	Logical level is completely static while entities belonging to physical level can migrate.
Fully mobile	Mobile	Mobile	Both levels can experience mobility. We are especially interested in this situation.

2 Mobility Taxonomy

Besides the mobile entities and the levels of mobility we identify types of mobility.

Physical mobility taxonomy

1. Based on how the connection is handled during migration
   (a) Discrete mobility: communication between entity and the rest of the environment goes through the following steps: (i) the communications halts; (ii) the entity moves; (iii) the entity connects in another location of the environment; (iv) the communication resumes;
   (b) Continue mobility: communication is alive during migration;
2. Based on the existence of a link between every mobile machine / user and a static machine
   (a) Tight mobility: there is a static machine for every mobile machine / user; the static machine is responsible for the interaction between the rest of the environment and its corresponding mobile machine / user;
   (b) Untight mobility: the above restriction is not imposed;
3. Based on connection / disconnection behavior
   (a) False mobility: the machine is connected / disconnected at the same location;
   (b) True mobility: the entity is disconnected from a location and connected to the other;

Logical mobility taxonomy

1. Based on the number of steps in physical mobile structure
   (a) One-step: the itinerary of logical entity contains one step in physical mobile structure;
   (b) N-steps: the above restriction is not imposed;

3 On Synchronization

Logical migration is conditional upon the existing physical links. Problems related to logical mobility in a mobile physical environment have as source modification of physical entities locations (e.g. connection / disconnection operations of machines / users). This urges the synchronization between logical entities migration and physical level mobility. We have identified two solutions for synchronization – passive and reactive behavior.

1. from physical level point of view
   (a) passive behavior of physical level towards logical level modifications: physical level checks periodically the existence of those logical entities that were prevented from migration (e.g. because the destination was disconnected). If there are such entities and if the actual physical level configuration allows their migration then it will be done;
   (b) reactive behavior of physical level towards logical level modifications: every time when a logical entity is prevented from migration because of physical level configuration, logical level notifies physical level. The outcome is that physical level configuration will be changed (immediately or with certain delay) so that logical migration will be possible;
2. from logical level point of view
   (a) passive behavior of logical level towards the physical level modifications: logical level checks periodically if physical configuration was changed so that logical migration is possible now;
   (b) reactive behavior of logical level towards the physical level modifications: every time when physical level changes, it notifies logical level. As soon as physical configuration allows, logical migration happens;

Mainly, passive behaviors consist in the following series: check, migration (if it is possible), sleep. The obvious disadvantage is a delay of logical entity migration. Despite it, in reactive behavior, as soon as it is required a modification of physical configuration (1b) or physical configuration allows logical migration (2b), the other level (physical respectively logical) is notified. We consider (2b) behavior more general than (1b). In order to describe our perception on a mobile environment with (2b)-like reactive behavior we resort to a mathematical description. The described environment is characterized by:

1. logical mobility: n-steps;
2. physical mobility: discrete, untight, true;
3. machines, users and logical entities as migratory items;

Because of the three directions of mobility (machine, user and logical entity) we call it 3-dimensional mobile environment ($3 - diMo$).

4 Formal Description

This mobile environment is defined as:

$$3 - diMo = (F, L),$$

where F represents physical level and L is logical level.

4.1 Physical Level

Physical level is given by:

$$F = (\mathbf{RM}, \mathbf{U}, C, \text{OpMigr}^p),$$

where $\mathbf{RM}$ is the set of real machines (mobile and static) belonging to the environment
$\mathbf{U}$ is the set of users (static or mobile) belonging to the environment
C is the multiset of connections between machines
OpMigr^p specifies migration at physical level

The set of real machine is defined as follows:

$$\mathbf{RM} = \{rm_1, rm_2, \ldots, rm_r\} = \mathbf{RM}_s \cup \mathbf{RM}_m,$$

where $\mathbf{RM}_s$ is the set of static real machines
$\mathbf{RM}_m$ is the set of mobile real machines

There are two categories of users, static (a static user works all time on the same machine) and mobile (migrate from a real machine to the other).

$$\mathbf{U} = \{u_1, u_2, \ldots, u_u\} = \mathbf{U}_s \cup \mathbf{U}_m,$$

where $\mathbf{U}_s$ is the set of static users
$\mathbf{U}_m$ is the set of mobile users

In order to deal with machines and users mobility as uniformly as possible we define the concept of *virtual machine*. Every time a user enters the $3-diMo$ environment (i) a virtual machine is created; (ii) a one-to-one association is created between the user and the virtual machine; (iii) a one-to-one association is created between the virtual machine and the real machine on which the user is actually working. When a user leaves the $3-diMo$, its corresponding virtual machine does the same. At a certain moment a user u_i works on a real machine rm_k and has a corresponding virtual machine mv_i. We can define the set of virtual machine:

$$\mathbf{VM} = \{vm_1, vm_2, ..., vm_u\} = \mathbf{VM}_s \cup \mathbf{VM}_m,$$

where vm_i is the virtual machine associated to the user u_i, $\forall\ i = 1, \ldots, u$ and $\mathbf{VM}_s$, $\mathbf{VM}_m$ correspond to $\mathbf{U}_s$ respective $\mathbf{U}_m$.

The following relation is true: $card(\mathbf{U}) = card(\mathbf{VM})$.

Let $\mathbf{M}$ be the set of all machines belonging to the environment:

$$\mathbf{M} = \{m_1, m_2, \ldots, m_n\}$$

We can have different points of view upon it, reunion of real with virtual machines or reunion of static (real and virtual) with mobile (real and virtual) machines.

$$\mathbf{M} = \mathbf{VM} \cup \mathbf{RM}$$

$$\mathbf{M} = \mathbf{M}_s \cup \mathbf{M}_m$$

where $\mathbf{M}_s = \mathbf{VM}_s \cup \mathbf{RM}_s$
$\mathbf{M}_m = \mathbf{VM}_m \cup \mathbf{RM}_m$.

Define the multiset C of connections between machines:

$$C = \{\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_n\}.$$

We have: $(\forall\ m_i \in \mathbf{M} \mid \exists\ \mathbf{C}_i \in C$ where $\mathbf{C}_i$ represents the connections of $m_i) \cap$ $(card(\mathbf{M}) = card(C))$

We group the connections between machines in two categories:

1. real connection: it is established between two real machines when a physical path exists between them;
2. virtual connection: it is established between a virtual machine and its corresponding real machine;

If $m_i \in \mathbf{RM}$ then
$\mathbf{C}_i = \{rm_k | (rm_k \in \mathbf{RM}) \cap$
(there is a physical connection between m_i and $rm_k)\} \cup$
$\{vm_k | (vm_k \in \mathbf{VM}) \cap$
(m_i is the real machine on which the user associated to vm_k
is working) $\}$
If $m_i \in \mathbf{VM}$ then
$\mathbf{C}_i = \{rm_k | u_z$ is the user associated to m_i and rm_k is the real machine
on which the user u_z is working $\}$

The following reciprocity relation is true:

$$(\forall\ m_i, m_k \in \mathbf{M})\ m_i \in \mathbf{C}_k \iff m_k \in \mathbf{C}_i$$

We define the physical world model as follows:

$$Model^p = (\mathbf{M}, <>^p),$$

where $<>^p\ :\ \mathbf{M} \times \mathbf{M} \longrightarrow \{0, 1\}$ is given by:

$$< m_i, m_k >^p = \begin{cases} 1, m_i \in C_k \\ 0, \text{ otherwise} \end{cases}$$

Reciprocity relation can be rewritten:

$$(\forall\ m_i, m_k \in \mathbf{M}) < m_i, m_k >^p = 1 \iff < m_k, m_i >^p = 1$$

Machines migration OpMigr^p is expressed by two class of operations, connection and disconnection of machines:

OpMigr^p= OpConn |
OpDisconn

We further refine these operations:

OpConn= OpConnRM |
OpConnVM

OpDisconn= OpDisconnTemp |
OpDisconnPerm

OpDisconnTemp= OpDisconnTempRM |
OpDisconnTempVM

OpDisconnPerm= OpDisconnPermRM |
OpDisconnPermVM

OpConnRM (rm_k) represents the connection of real machine rm_k:

Pre-conditions
$(rm_k \in \mathbf{RM} \cap \mathbf{C}_k = \emptyset) \cup rm_k \notin \mathbf{M}$

Operation description
begin
  $\mathbf{C}_k = \emptyset$
  for every $rm_i \in \mathbf{RM}$
  begin
    if (there is a physical path between mr_i and mr_k) then
    begin
      $\mathbf{C}_i = \mathbf{C}_i \cup \{rm_k\}$
      $\mathbf{C}_k = \mathbf{C}_k \cup \{rm_i\}$
    end
  end
  if $rm_k \notin \mathbf{M}$ then // *is not the case of a reconnection after a temporary*
                    // *disconnection*
  begin
    $\mathbf{M} = \mathbf{M} \cup \{rm_k\}$
    $C = C \cup \mathbf{C}_k$
  end
  notify LMSS
end

Post-condition
Reciprocity relation is true

OpConnVM (vm_k, rm_i) represents the connection of virtual machine vm_k to real machine rm_i:

Pre-conditions
$(vm_k \in \mathbf{VM} \cap \mathbf{C}_k = \emptyset) \cup vm_k \notin \mathbf{M}$

Operation description
begin
  $\mathbf{C}_i = \mathbf{C}_i \cup \{vm_k\}$
  $\mathbf{C}_k = \mathbf{C}_k \cup \{rm_i\}$
  if $vm_k \notin \mathbf{M}$ then // *is not the case of a reconnection after a temporary*
                    // *disconnection*
  begin
    $\mathbf{M} = \mathbf{M} \cup \{vm_k\}$
    $C = C \cup \mathbf{C}_k$
  end
  notify LMSS
end

Post-condition
Reciprocity relation is true

When a machine is temporary disconnecting it is supposed that after a while it will connect again. This assumption cannot be made for a permanently disconnection operation.

OpDisconnTemp RM(rm_k) represents the temporary disconnection of real machine rm_k :

Pre-conditions
$rm_k \in \mathbf{RM} \cap \mathbf{C}_k \neq \emptyset$

Operation description
begin
  for every $rm_k \in \mathbf{RM}$
  begin
    if $rm_k \in \mathbf{C}_i$
    begin
      $\mathbf{C}_i = \mathbf{C}_i \backslash \{rm_k\}$
      $\mathbf{C}_k = \mathbf{C}_k \backslash \{rm_i\}$
    end
  end
end

Post-condition
Reciprocity relation is true

OpDisconnTempVM(vm_k, rm_i) represents the temporary disconnection of virtual machine vm_k from the real machine rm_i:

Pre-conditions
$vm_k \in \mathbf{VM} \cap \mathbf{C}_k \neq \emptyset \cap rm_i$ is real machine on which the user associated to vm_k is working

---

Operation description
begin
$\mathbf{C}_i = \mathbf{C}_i \backslash \{vm_k\}$
$\mathbf{C}_k = \mathbf{C}_k \backslash \{rm_i\}$
end

Post-condition
Reciprocity relation is true

OpDisconnPermRM (rm_k) represents the permanent disconnection of real machine rm_k :

Pre-conditions
$rm_k \in \mathbf{RM} \cap \mathbf{C}_k \neq \emptyset$

Operation description
begin
for every $m_i \in \mathbf{M}$
begin
if $rm_k \in \mathbf{C}_i$
begin
$\mathbf{C}_i = \mathbf{C}_i \backslash \{rm_k\}$
$\mathbf{C}_k = \mathbf{C}_k \backslash \{rm_i\}$
end
end
$\mathbf{M} = \mathbf{M} \backslash \{rm_k\}$
$C = C \backslash \mathbf{C}_k$
end

Post-condition
Reciprocity relation is true

OpDisconnPermVM(rm_k, rm_i) represents the permanent disconnect operation of virtual machine vm_k from reale machine rm_i :

Pre-conditions
$vm_k \in \mathbf{VM} \cap \mathbf{C}_k \neq \emptyset \cap rm_i$ is real machine on which the user associated to vm_k is working

---

Operation description
begin
$\mathbf{C}_i = \mathbf{C}_i \backslash \{vm_k\}$
$\mathbf{C}_k = \mathbf{C}_k \backslash \{rm_i\}$
$\mathbf{M} = \mathbf{M} \backslash \{vm_k\}$
$C = C \backslash \mathbf{C}_k$
end

Post-condition
Reciprocity relation is true

Pre–conditions associated to every operation ensure a correct succession for connection and disconnection operations. If we consider this succession from one machine point of view then after a connection operation can follow a disconnection one and vice-versa. If we analyze from the environment point of view as a whole, two or more connection / disconnection operations can go one after the other.
OpMigr^p modifies physical world model.

4.2 Logical Level

Logical level is given by:

$$L = (\mathbf{LE}, \mathbf{LMSS}, \text{OpMigr}^l),$$

where **LE** is the set of logical entities belonging to the environment
LMSS is Logical Mobility Support Service
OpMigr^l specifies logical level migration

LMSS is the set of entities preventing from migration because source / destination is disconnected; this component implements a reactive behavior of logical level to the physical configuration modifications; it is a way to manage the entities that are prevented from migration at a certain moment of time.

There are two types of logical entities working within the environment:

1. static: a static entity works all its life span on the same machine;
2. mobile: migrate from one machine to the other;

We define logical entities **LE** set

$$\mathbf{LE} = \{le_1, le_2, \ldots, le_l\}$$

Let $\mathbf{LE}_{tr}$ be the set of logical entities being in transit at a certain moment t. The following relations are true at the moment t :

$$\mathbf{LE}_1 \cap \mathbf{LE}_2 \cap \ldots \cap \mathbf{LE}_n \cap \mathbf{LE}_{tr} = \emptyset$$

$$\mathbf{LE}_1 \cup \mathbf{LE}_2 \cup \ldots \cup \mathbf{LE}_n \cup \mathbf{LE}_{tr} = \mathbf{LE}$$

where $\mathbf{LE}_i$ represent the set of logical entities running on machine m_i.

We can see **LE** as a reunion between static logical entities $\mathbf{LE}_s$ and mobile logical entities $\mathbf{LE}_m$:

$$\mathbf{LE} = \mathbf{LE}_s \cup \mathbf{LE}_m$$

We define logical world model as follows:

$$Model^l = (\mathbf{M}, <>^l),$$

where $<>^l$: $\mathbf{LE} \times \mathbf{M} \longrightarrow \{0,1\}$ is given by

$$< le_i, m_k >^l = \begin{cases} 1, le_i \in \mathbf{LE}_k \\ 0, \text{otherwise} \end{cases}$$

Logical entity checks the existence of a physical path between source and destination. If this condition is fulfilled the entity migrates, otherwise **LMSS** takes it under its control. From now on it is physical entities responsibility to notify **LMSS** about every connection.

OpMigr$^1(le, m_{surs}, m_{dest})$ represents migration of le from m_{surs} to m_{dest} :

Pre-conditions
$le \in LE_{surs}$

Operation description
begin
if $(m_{surs} \in \mathbf{RM} \cap m_{dest} \in \mathbf{RM} \cap < m_{surs}, m_{dest} >= 1) \cup$
$(m_{surs} \in \mathbf{VM} \cap rm_k \in \mathrm{C}_{surs} \cap m_{dest} \in \mathbf{RM} \cap < rm_k, m_{dest} >= 1) \cup$
$(m_{surs} \in \mathbf{RM} \cap m_{dest} \in \mathbf{MV} \cap rm_k \in \mathrm{C}_{dest} \cap < m_{surs}, rm_k >= 1) \cup$
$(m_{surs} \in \mathbf{VM} \cap rm_k \in \mathrm{C}_{surs} \cap m_{dest} \in \mathbf{VM} \cap rm_i, \in \mathrm{C}_{dest} \cap$
$< rm_k, rm_i >= 1)$
begin atomic op *//there is a physical path between* m_{surs} *and* m_{dest}
$\mathbf{LE}_{dest} = \mathbf{LE}_{dest} \cup \{le\}$
$\mathbf{LE}_{surs} = \mathbf{LE}_{surs} \backslash \{le\}$
end
else *//migration is not possible so* ***LMSS*** *is informed*
begin
$\mathbf{LMSS} = \mathbf{LMSS} \cup \{(m_{surs}, m_{dest}, le)\}$
end
end

Post-condition
$el \in \mathrm{EL}_{dest}$

For every connection notification an inspection is performed in order to detect those logical entities whose migration is now possible.

Pre-condition
m_n sends a connection notification

Operation description
begin
 begin
  for every waiting logical entity (m_{surs}, m_{dest}, le)
   if $(m_{surs} = m_n \cup m_{dest} = m_n) \cup$
    $(m_{surs} \in \mathbf{RM} \cap m_{dest} \in \mathbf{RM} \cap < m_{surs}, m_{dest} >= 1) \cup$
    $(m_{surs} \in \mathbf{VM} \cap rm_k \in \mathrm{C}_{surs} \cap m_{dest} \in \mathbf{RM} \cap < rm_k, m_{dest} >= 1) \cup$
    $(m_{surs} \in \mathbf{RM} \cap m_{dest} \in \mathbf{RM} \cap rm_k \in \mathrm{C}_{dest} \cap < m_{surs}, rm_k >= 1) \cup$
    $(m_{surs} \in \mathbf{VM} \cap rm_k \in \mathrm{C}_{surs} \cap m_{dest} \in \mathbf{VM} \cap rm_i \in \mathrm{C}_{dest} \cap$
    $< rm_k, rm_i >= 1)$
   begin atomic op *//migration is possible because there is a physical path*
                //between m_{surs} *and* m_{dest}
    $\mathbf{LE}_{surs} = \mathbf{LE}_{surs} \backslash \{le\}$
    $\mathbf{LE}_{dest} = \mathbf{LE}_{dest} \cup \{le\}$
    // remove logical entity from ***LMSS*** *management*
    $\mathbf{LMSS} = \mathbf{LMSS} \backslash \{(m_{surs}, m_{dest}, m_{le})\}$
   end
 end
end

Post-condition
not exists $(m_i, m_j, le) \in \mathbf{LMSS}$ so that $((m_i = m_n \cup m_n = m_j) \cap$ there is a path between m_i and $m_j)$

We define world model:

$$Model = (Model^f, Model^l)$$

5 Overview of Existing Mobile Agent System

We discuss how the functionality of the existing mobile agent systems carries on in a mobile physical environment and how they addresse the synchronization problem.

One of the systems that deal with physical mobility yet in design phase is Gypsy [15]. A typical scenario for synchronization between logical and physical level is one in which the user working on a mobile device launches an agent that migrates through the static part of the network with the aim of gathering information. If when the agent tries to come back to the owner his machine is disconnected the agent will be stored at user place on the home server waiting to be retrieved. Depending on the notification policy the user will be informed using email or SMS messages about the fact that the agent has returned. The user reconnects and asks home server to forward the waiting agent. In [2] the

same technique is used for logical and physical level synchronization, the user launches an agent from a mobile device and then it disconnects. After gathering the results the agent tries to go back to the user's device. If this is not yet connected the agent either waits at an Agent Meeting point for a user's sign or it alerts the user sending a message. All these scenarios belong to reactive behavior of physical level towards logical level modifications according to above classification.

Another mobile agent platform that provides support for physical mobility is D'Agents [3]. Its docking system associates to every laptop a static dock machine, that will carry on the whole communication of the laptop. D'Agents implemented a reactive behavior of logical level to physical configuration changes.

Apart from the above-mentioned systems, there are mobile agent platforms that although not even address physical mobility, yet are capable of handling in a rudimentary way the synchronization problem. One of them is Aglets [4] that provides *retractAglet* one may use to implement passive behavior. A scenario might be one in which an agent is prevented from returning to its disconnected owner. The agent could stay at the current place till the owner reconnects and pulls the agent back.

It is worth mentioning the three categories of mobile agent systems identified: (i) those that were designed to work properly in a mobile physical environment; (ii) those that even if were not designed with physical mobility as an issue still handle it; (iii) those that are not able to cope with it. All discussed systems provide explicitly support for machines and logical entities mobility but not for users mobility.

6 Conclusions and Further Work

Our aim was to tackle mobile environments from three points of view (i) levels of mobility; (ii) entities that feature mobile behavior and types of mobile behaviors; (iii) the way logical and physical levels deal with each other.

We strongly believe that in order to work out a problem one has to go through the following steps: (i) identification of possible solutions; (ii) analysis of solutions and choice of one of them; (iii) (formal) description of the selected solution; (iv) designing and implementation. The problem we are dealing with is the development of a mobile environment with machines, users and logical entities as migratory components and with synchronization between logical and physical mobility. In this paper we identified the possible solutions – passive and reactive behavior, chose a solution and described it. Apart from all these we presented an original taxonomy for mobility and discussed a survey of existing mobile agent systems from synchronization viewpoint. The two types of behavior – passive and reactive – could be used as classification criterion for mobile agent systems. We consider formal description a starting point for a future designing of a infrastructure that smoothly synchronizes logical and physical level.

References

1. Carzaniga A.; Picco G.P. and Vigna G.: Designing Distributed Applications with Mobile Code Paradigms, In Proceedings of the 19th International Conference on Software Engineering (ICSE'97), Boston USA, ACM Press, p.22-23, May 1997
2. Chess, D.; Grosof, B.; Harrison, C.; Levine, D.; Parris, C. and Tsudik, G.: Itinerant Agents for Mobile Computing, Readings in Agents, Morgan Kaufman, 1998
3. Gray, R.; Kotz, D.; Nog, S.; Rusu, D. and Cybenko, G.: Mobile Agents for Mobile Computing. Technical Report PCS-TR96-285, Department of Computer Science Dartmouth College, May 1996
4. Lange D.B. and Oshima M.: Programming and Deploying JavaTM Mobile Agents with Aglets, Addison-Wesley, 1998
5. Lugmayr W.: Gypsy: A Component-oriented Mobile Agent System, Dissertation for Dr. Techn., Technical University of Vienna, October 1999
6. Maass, H.: Location-aware mobile applications based on directory services, Mobile Networks & Applications, 3(2), p.157-173, August 1998
7. Mascolo C.: Specification, Analysis, and Prototyping of Mobile Systems, In Doctoral Symposium of the 21st International Conference on Software Engineering, Los Angeles, IEEE/ACM Press, May 1999
8. Mascolo C.: MobiS: A Specification Language for Mobile Systems, In Third International Conference on Coordination Models and Languages, Amsterdam, The Netherlands, LNCS 1594, Springer-Verlag, p.37-52, April 1999
9. Picco, G.P.: Understanding, Evaluating, Formalizing, and Exploiting Code Mobility, Ph.D. Thesis, Torino Polytechnic, 1998
10. Picco, G.P. and Roman G.C.: Developing Mobile Applications with Lime, Technical Report WUCS-00-05, Washington University, USA, February 2000
11. Rarau, A.; Salomie, I. and Pusztai, K.: Agent-Based Mobile Environment, In Proceedings of Workshop 2000 Agent-Based Simulation, p.221-225, May 2000
12. Sahai, A. and Morin, C.: Mobile Agents for Enabling Mobile Users Aware Applications, In Proceedings of the Second International Conference on Autonomous Agents, p.205-211, May 1998
13. Satoh I.: A Formalism for Hierarchical Mobile Agents, In Proceedings of International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE' 2000), IEEE Press, June 2000
14. Villate Y.; Gil D.; Goni A. and Illarramendi A.: New Challenges for Mobile Computers: Combinations of Indirect Model and Mobile Agents, In Proceedings of ICSE98 International Workshop on Computing and Communication in the Presence of Mobility, Kyoto (Japan), July 1998
15. http://www.infosystem.tuwien.ac.at/Gypsy

YAAP: Yet Another Active Platform

Nicolas Rouhana[1], Samer Boustany[1], Eric Horlait[2]

[1] University Saint-Joseph – Lebanon
Nicolas.Rouhana@usj.edu.lb
Samer.Boustany@fi.usj.edu.lb
[2] University Pierre et Marie Curie – France
Eric.Horlait@lip6.fr

Abstract. We present YAAP: a generic active network architecture with the ability to manually and dynamically deploy network services within distributed network nodes and providing a secure execution environment (EE). It also demultiplexes active network packets to multiple EEs located on the same network node. Currently, YAAP prototype is implemented under the Linux operating system, with some parts built in kernel-space and others in user-space to optimize performance. This paper describes the node architecture, serves as an internal description of the implementation, and also demonstrates the utility of YAAP's activeness by experimentally deploying active applications using the current implementation.

1 Introduction

Active networks [1] represent today a "quantum step" towards the evolution of legacy networks to render network nodes more programmable, resulting in a much rapid and dynamic way for protocol and service deployment. By providing a *programmable* interface in network nodes, they expose the underlying resources and functionality, and provide mechanisms for constructing or refining new services from those elements. In active networks, code mobility represents the main vehicle for program delivery, control and service construction. Network behavior can hence be controlled on a per-packet, per-user, or other basis. The network is seen as an API defining a *virtual machine* as well as a per-node packet processing behavior, and the code through which users control that behavior is via the packets they send.

In that respect, several active networks platforms have recently emerged focusing on a specific "active" application, i.e. protocol deployment with ANTS [2], network management with Smart Packets [3], network service deployment with Anetd [4], and so forth. The "granularity of control" [5], which refers to the scope of a switch/router behavior being modified by a received packet to accomplish the "active" functions, is different form one platform to another. At one extreme, called the discrete approach, a single packet could boot a complete software environment seen by all packets arriving at the node. At the other extreme, called the integrated approach, a single packet (e.g., a "capsule" [2]) can modify the behavior seen only by that packet.

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 185-194, 2000.

A number of pertinent questions always arise in each case to find out the most flexible design of an active network and the most "killer" active applications. First, what kind of approach should be used, integrated or discrete, to implement the active network, and what are the necessary components to apply into active nodes? Second, how current legacy networks can be seamlessly "upgraded" to active networks without compromising sensitive issues such as interoperability, performance and safety? Finally, how can users and service providers benefit from the new network's functionalities on a large-scale and wide-area basis?

The current existing active platforms already mentioned try to tackle some of the issues while overlooking others, thus limiting their wide spread use and deployment. For instance, most platforms have poor "real life" performance since they execute in user-space and present a "virtual" execution environment based on pre-configured point-to-point connectionless UDP channels to overlay the active network on top of the legacy network, which constitutes also a configuration burden. Most platforms are application-specific which limits the functionalities they offer.

In this article, we present YAAP, an active network platform – inspired by the existing platforms – defining a generic architecture for an active node that can easily support what are now considered the most needed "active" functionalities within an active node, namely network management, distributed service deployment, and dynamic execution of functions, without compromising security and performance. The next section describes the architecture of a YAAP node and its internal modules. Section 3 details the implementation of the dispatcher module of YAAP. Section 4 shows how the services are implemented whilst section 5 gives an example of a service deployment using YAAP. We present future works and conclusions in sections 6 and 7 respectively.

2 YAAP Architecture

In order to achieve our goals, the YAAP node consists of four parts, as shown in figure 1: (1) an encapsulation scheme for data delivery between nodes, (2) a service application part delivering the three native "active" services mentioned in section 1 (network management, distributed service deployment, and dynamic execution of functions) or even any Execution Environment (EE), (3) a dispatcher that forwards the "active" packets towards the required service application, and (4) a security model.

Currently, YAAP nodes implement the dispatcher module and service deployment module using Linux OS, with parts of the architecture at kernel level and other parts at user-level for the sake of performance. Also, in order to provide a safe computation environment, every node implements a java virtual machine in which the services run, and that makes use of Java Native Interface (JNI) to establish communication between the virtual machine in user-space and the kernel.

The next sub-sections detail the role of the parts shown in figure 1.

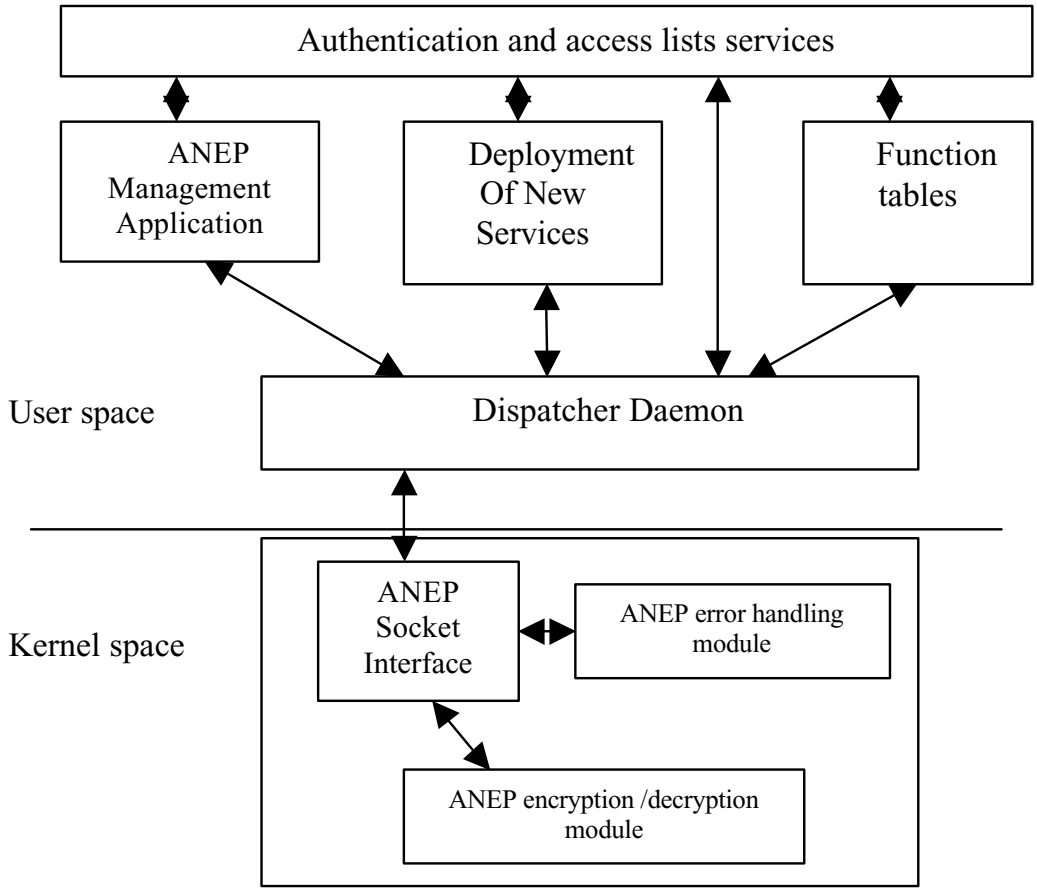


Fig. 1. YAAP node architecture

2.1 Packet Transmission

To insure interoperability with existing networks, YAAP uses direct ANEP [6] in IP encapsulation for the transmission of the "active" packets. ANEP has been implemented in kernel-space along side IP to provide faster response time. We use for that purpose a new protocol code specifying the ANEP protocol in the IP header to demultiplex packets at the IP level. The format of the ANEP packet is given in figure 2.

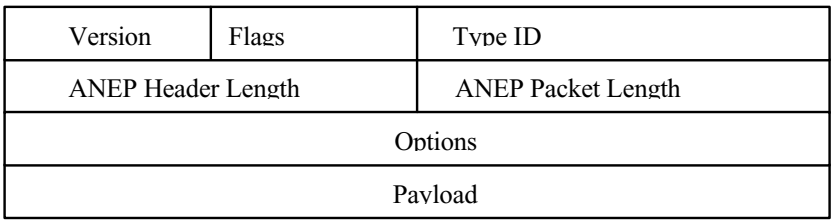


Fig. 2. Format of ANEP packets

The typeID is used to demultiplex the Execution Environment (or service) within the YAAP active node. We use the MSbit of the Flags field to 0 to allow the packet to be

"executed" locally. We had to make intermediate YAAP routers evaluate each ANEP packet even though it is not the final destination. A minor modification was done to the IP routing code in Linux allowing the kernel to deliver the packet - when the specified ANEP flag bit is set - to YAAP. Furthermore, an API developed for YAAP (called YAPI) allows users to seamlessly create ANEP packets the same way they would with TCP/UDP API (i.e. sockets) and which is used by the services. An error-handling module interacts with the ANEP socket interface and handles the platform's specific errors. Also, an encryption/decryption module secures the communications.

2.2 The Dispatcher

The dispatcher daemon module constitutes the interface between the kernel and the different components of the node and also acts as a "bus" enabling the different components to communicate with each other as seen in figure 1. The dispatcher operations are detailed in section 3.

2.3 The Application Modules

Three application modules are defined in YAAP that are considered the most used "active" applications today. The first application, the management application service module in YAAP, follows classical network management standards, but adds "active" functionalities allowing a network administrator for instance to write a program in Java that accesses the node's MIB to read or write specific variables. Provided this interface to the MIB, the manager sends the specific program encapsulated in ANEP to the managed node where it is evaluated. Not only the manager will be able to perform complex operations on MIB variables, but can also for example download a "watch dog" program that takes user-specified actions in critical situations and report the results to a specific node. Furthermore, the interface to the MIB can be easily expanded, making it possible to the manager to add dynamic objects to the MIB.

The deployment of new services module in YAAP permits the loading of new software at the node, and has three major functions. First, it makes the existing services on its node available for other nodes. Second, when a new service is needed, it contacts the node where the service resides, downloads and installs it on the node by creating the necessary entries in the dispatcher. The scheme is explained in section 4 and uses a discrete approach. Finally, this component can upgrade the Management Application to enable it to manage the newly installed service. A naming scheme, similar to URLs [4], could also be used to locate the resource that contains the required service.

The third application component in YAAP allows a number of functions to be downloaded into the nodes allowing specific computations to be done on packets (e.g. changing the encoding of a video stream). It can also be used for instance in the case of several services needing a same set of functions from within the network; instead of redefining the functions, the services can reference the specific functions. To have

a packet processed by a particular function in the node, an option in the packet's header indicates which function to execute. The naming of the functions follows the same rules established for downloading new services.

2.4 The Security Module

The fourth part of the architecture is the security module, used by the above three application modules for authentication and access to the node's execution environment. For instance, the management component must know who can read or modify the MIB, add objects, etc. The new services component must know who can access or download a particular service and must also determine whether a needed service should be downloaded or not. Finally, the downloaded functions table needs to know who can access the required functions.

3 The Dispatcher

The dispatcher is the heart of the YAAP node, and has several functions:

- The initiation of the download mechanism of a service
- Local registration of a service and its creation
- Registration of a client for a given service

The next subsections detail each of the functions.

3.1 Initiation of the Download Mechanism of the Service

As already stated in section 1, all the existing platforms listen on a given UDP port to demultiplex the corresponding service. Therefore, whatever the Type ID of an active service, the kernel will always have the possibility to deliver the packet to the node (via the UDP port); the node will then check if it supports the received Type ID. If not, it will initiate the mechanism that allows it to download the service (as in ANTS for instance with demand-loading [2]). But in our case, considering that ANEP is implemented in the kernel, and considering that each active service will listen on its own ANEP Type ID, the solutions adopted until now by all the existing active platforms are not satisfactory. To illustrate this fact, let us take the following example: suppose an ANEP packet with a given Type ID arrives to our node; the kernel receives it first, then consults its sockets hash table to see if an existing socket is listening on this Type ID. If an entry is not found, the kernel usually discards the packet and the dispatcher will not be aware of this incoming packet. To overcome this problem, several changes were made to the kernel source code:

- The function that delivers the packet to the socket should be modified not to discard the packet when a corresponding socket is not found, but try instead to queue the packet in a classical message queue (used habitually for inter-process

communication). In this case, the kernel acts as a client to the message queue. This supposes that the dispatcher itself creates this queue. Thus, the YAAP node has to be launched, or else the kernel will not be able to have a reference to this queue (that has 192977732 as key) and will therefore discard the packet.

- The message-based inter-process communication system was conceived for the communication between two processes and not between the kernel and a process. To overcome this minor limitation, two functions were added in the *msg.c* file of the kernel: the first, `sys_sndmsg_from_kernel()` which is similar to the original `sys_sndmsg()` but without checking the User ID and Process ID (since they do not exist), and calls the second new function `load_msg_from_kernel()` instead of calling the old `load_msg()` function. The `load_msg_from_kernel()` function is similar to the original `load_msg()` but uses the `memcpy()` function instead of the `get_from_user()` function since the data that will be placed in the message queue is already in the kernel space and does not need to be copied from user space segment to the kernel space segment.

To summarize, the dispatcher creates a message queue having 192977732 as key, and will be notified through this queue by the kernel whenever a packet having a non-supported Type ID arrives at the node. Furthermore, the kernel hands to the dispatcher the Type ID of the packet, the source address and the destination address and the data contained in this packet. Thus the dispatcher will be able to initiate the download mechanism of the new and unknown service that will further process the packet.

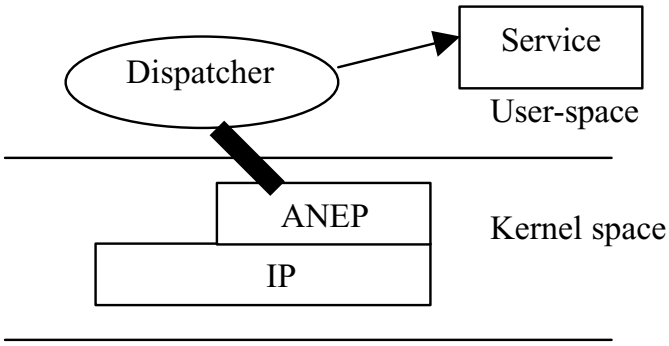


Fig. 3. Kernel-user space communication in YAAP

After downloading the service (through a mechanism explained later), the dispatcher creates a process that will do the following operations:

- Creation of a message queue, which will be used by the clients.
- Creation of a java virtual machine using the JNI (Java Native Interface).
- Creation in the virtual machine of an object of the class `AnepClassLoader` that will be used for the definition of the service class and the creation of an object of this class.
- Definition of the service class and creation of an object of this class passing to it the key of the created message queue.

- Creation of an ANEP socket with the Type ID of the new service
- Creation of the thread that will wait for messages on the message queue.

All these operations, showed together in figure 4, will be further detailed later.

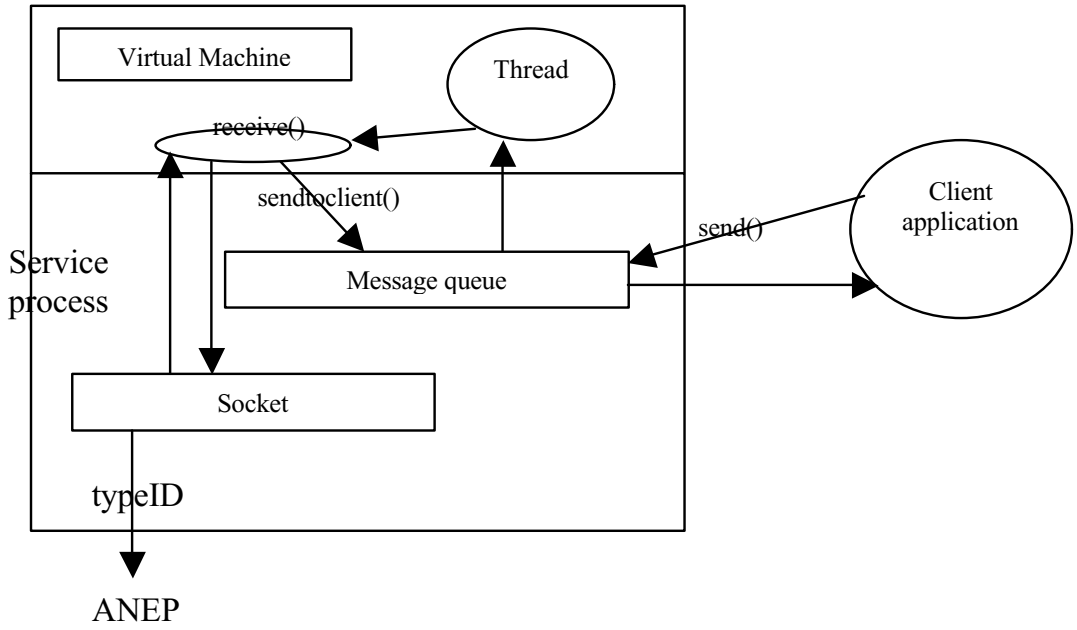


Fig. 4. Architecture of a service at the node

3.2 Manual Registration and Creation of a Service

The YAAP node must also allow the "manual" registration of services. A specific application, *registerservice*, has been developed for that purpose. Initially, services can exist on the local machine or on a distant machine. In either case, the *registerservice* application can be used with one of two options: a first option specifies that the service is located on the local machine and hence a file name must be given as argument, whilst the other option specifies that the service is located on a distant machine and hence the name of the machine as well as the name (or the Type ID) of the service should be given as arguments.

With the first option, the file that is given as argument must contain the name of the service class, its Type ID and the location of the file of the class; followed by the different classes of the PDUs used by this service and the location of their respective files. The *registerservice* application then sends a message (with an ID equal to 2) to the dispatcher containing the file name through the initial message queue (the one that has 192977732 for key) to inform it that a service must be registered. Having received the file name, the dispatcher will read the files of the specified classes and will then effectuate the same operations described in subsection 3.1.

With the second option, (when the service is located on a distant node), *registerservice* sends a message (with an ID equal to 3) to the dispatcher containing

the name of the machine and the name or the Type ID of the service to be downloaded, through the initial message queue. Having received this information, the dispatcher passes the hand to the deployment service (which will be explained later).

4 Services and PDUs

For new service deployment, the approach adopted is the discrete approach, for the simple reason that we believe that not all services require evaluation at intermediate nodes (a *ping* for instance). Furthermore, in order to enforce a safe and secure execution environment, all services are written in Java, and the JNI provides communication between the written java classes and the processes created by the dispatcher.

The PDUs encapsulated in ANEP and sent to the nodes are Java classes without any functions. These classes are formed exclusively by data members entirely "serializable" implementing the `Serializable` interface. Furthermore, they must contain at least four fields: source address, destination address, client ID, and a flag to indicate whether the PDU should be delivered to the client or not.

All the services require the following functions:

- `receive()`: called by the service process, when it receives an ANEP packet with the corresponding typeID of the service, which hands the data as well as the source address of the packet to the corresponding service. It uses the `readObject()` function to "deserialize" the received PDU and get its corresponding class.
- `send()`: is called to put the PDU in the message queue of the corresponding service. It uses `writeObject()` to "serialize" the PDU object before sending it.
- `sendtoclient()`: used to pass the PDU from the virtual machine to the message queue of the process to be evidently received by the client.

4.1 Registration of a Client for a Given Service

Each client on a node must be able to use a particular service on that node, i.e. sending and receiving PDUs of this service. Therefore, the client should go through a registration phase within the node to be able to use the service. The method used is the following: each client sends to the dispatcher through the initial message queue a message having an ID 4 specifying the Type ID of the service it wishes to use, then blocks while awaiting for a response. Having received this message, the dispatcher replies to the client giving him a unique number and the key of the message queue created at the time of the creation of the service. The client then uses that number in all the PDU that it emits. To send a PDU, the client calls a `send()` method (through the JNI) that uses the key passed by the dispatcher to know which message queue to use. To receive a PDU, the client blocks itself by calling a `receive()` method that

waits for a message having the client number as MessageID to appear in the service message queue.

5 Example

After explaining the above mechanisms, we now illustrate how to deploy a simple *ping* service called `AnepPing` with typeID 15 that only uses one PDU called `AnepPingPDU`, and we suppose having only one client called `AnepPingClient`.

The `AnepPingPDU` class contains an additional data member called `count`, an integer to be incremented at the destination node. The `receive()` function of the `AnepPing` class deserializes the arriving data, casts the object to an `AnepPingPDU`, and tests the *deliver* flag. If the flag is false, and the source address in the PDU is different that the local node, `count` is incremented, the *deliver* flag is set to true and a `send()` is called to the node having the source address. If however the flag is false and the source address in the PDU equals the local node address, a `send()` is called to the node having the destination address. Finally, if the *deliver* flag is true, the `receive()` function calls `sendtoclient()`.

We now consider two YAAP nodes N1 and N2 with the above classes residing in N1. The different steps to distribute the *ping* service between the two nodes (in a Linux environment) are as follow:

- The service is registered with the application *registerservice*. A process is then created with a socket descriptor having typeID 15, a key for a message queue, and an object of class `AnepPing` is created within a java virtual machine.
- The client is started using the command *java AnepPingClient.* The client sends a message to the dispatcher to get the message queue key and a unique number.
- The client builds an object of type `AnepPingPDU`, resets the values of `count` and *deliver* flag to 0, serializes the object and sends it to the message queue of the process that created the `AnepPing` service.
- The PDU is read from the message queue by the process and sent to its destination (N2).
- The dispatcher at N2 is notified of the packet arrival with a new typeID. N2 stores the packet and downloads the service.
- After downloading, the packet is handed to the process of the `AnepPing` service in N2 which calls the `receive()` function of the `AnepPing` object running in the virtual machine. The `receive()` functions executes and the PDU with a `count` value to 1 is passed to the message queue of the process which created `AnepPing`.
- The PDU is pulled from the message queue by the process and sent to its destination (N1).
- The PDU is received by the socket at N1 which listens to typeID 15. Again, the `receive()` function of the `AnepPing` object is called. The deliver flag being

true, the PDU is handed to the process message queue with a MessageID equal to the number given to the client.

- The client, which is waiting to receive a message with an ID equal to the list number of the key got from the dispatcher (and that corresponds to the key of the message queue of the process of `AnepPing`), receives the PDU, deserializes it and can now verify the value of `count`.

6 Future Works

Having implemented the core engine of our YAAP architecture, namely the packet transmission scheme, dispatcher and the dynamic service deployment, we are now focusing our efforts on building the management module, which is an integrated-based approach. The PDUs of this service carry functions which would call existing network management primitives inside the nodes for MIB browsing and modification. This service would necessary make use of the security schemes to provide safety access, so the security module is also being analyzed in depth.

7 Conclusions

We presented our vision of an architecture for an active node that is able to handle a certain amount of "active" applications in a secure execution environment without sacrificing performance. Our work so far leads us to believe that this is a very promising approach to active network design, and the implementation is available upon request for research purposes.

References

[1] D. L. Tennenhouse and D. J. Wetherall. "Towards an Active Network Architecture." Computer Communication Review, Vol. 26, No. 2, April 1996.

[2] D. J. Wetherall. "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols". Submitted to ACM SIGCOMM'97, Cannes, France. September 1997.

[3] C. Partdridge and A. Jackson. Smart packets. Technical report, BBN, 1996. http://www.net-tech.bbn.com/smtpkts/smtpkts-index.html.

[4] L. Ricciulli. "Anetd: Active NETworks Daemon (v1.0)". http://www.csl.sri.com/ancors.

[5] Calvert, K. et. al, "Directions in Active networks", *IEEE Communications Magazine, Special Issue on Programmable Networks,* October 1998.

[6] S. Alexander, C. A. Gunter, A. D. Keromythis, G. Minden and D. Wetherall. ANEP: Active Network Encapsulation Protocol. http://www.cis.upenn.edu/switchware/ANEP/docs/ANEP.txt, 1997.

Searching for Music with Agents

Natasha Kravtsova, Andre Meyer

n.kravtsova@tue.nl, kravtsov@research.philips.com
andre.meyer@philips.com

Abstract. This paper describes the application of mobile agents for searching music files. The focus of the work was on seeking the best solution to search for music meta-data by using mobile agent technology. Both user requirements and technological side of agents has been addressed. The system that has been developed is purely distributed with no central server for organisation and communication. Agents, existing on users' computers, solve all questions of communication and organization without user interaction. When using the system, user should adhere to the following behavioural rule: you want to use the system, you should provide some resources of your computer for agents of other users. The first prototype was built using Java and Voyager ORB.

1. Introduction

Agent technology is becoming more widespread as the availability of network access and the demand for the use of agents become greater. Electronic commerce and the Web are a big attraction for developers to design agents that can make behind-the-scenes work cheaper, faster or more effective. The Internet is made up of many millions of computers, some of which are permanently connected, but most of the connections exists only for short periods of time (via dial-up modem connections). Imagine if you could delegate some tasks to mobile agents that would roam the network for you while you are not connected. This goal would be extremely desirable in the short period, until permanent connections became more prevalent.
As the number of portable digital devices grows, wireless networks that allow for flexible, timely and efficient data communication become more and more important.
A mobile agent is not bound to the system where it started execution. It has the unique ability to transport itself from one system in the network to another. The ability to travel allows a mobile agent to move to a system that contains an object with which the agent wants to interact and then to take advantage of being in the same host or network as the object [1].

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 195-203, 2000.

This project aims at developing a concept to search and share music files among users on the Internet, based on agent theory. The concept includes the development of new ways of data processing, the development of an agent-based architecture of the system and the user-agent interaction model.
The following question was posed: To what extent can agent theory be used for searching music files out the Internet. To answer this question, the research has pursued in two main directions: defining the user requirements and the technological requirements regarding agent technology.

- *User requirements*. The interest of the study was to uncover the way in which people search for music on the Internet. For instance what kind of information do they use for search (keywords), what do they like and dislike during the search. This information was used to improve the existing ways of search and to use it in the implementation of music agents.
- *Technological requirements*. The most common problem for all existing agent systems is the lack of an infrastructure, that is, a platform on which mobile agents need to exist and function.

At the same time, many resources exist that provide music information for users. However, it is too time-consuming and cumbersome to search for this information manually, when it is timely needed.

2. User Requirements

The purpose of the music agents is to help users to find the music meta-data on the Internet. We will present requirements, which concern two main directions: user-agent communication and music search.

User-Agent communication. Our interest was mainly threefold. First, how would people like to search for music files. Second, what keywords are were often used. Third, what is the user attitude towards the agent idea.
According to the results of a questionnaire and studies that were done before [2], we can draw up the following user requirements.

- Effective communication between users and their agents strongly depends on a meaningful and accurate agent representation (*metaphor requirements*).
- Even though agents are intended to perform tasks in a relatively autonomous way, the user should delegate these tasks explicitly. The user should be able to customize agent functionality as well as assign, suspend, resume and cancel agent tasks at any time (*user control requirements*).
- The autonomous nature of agent operation raises concerns regarding data integrity, user privacy, and agent dependability (*security requirements*).
- If agents are to be useful, they should be able to adapt to the often dynamic needs and preferences of the user. Means should be provided for agents to represent and maintain knowledge about the user, and techniques are needed to support adaptive behaviour (*adaptivity requirements*).

Music search. In this paper only search requirements are shown, which we will try to fulfil:

- *decrease the time which the user spends for search* (Uses commented that the search procedure takes much time. They want this time to be decreased. Currently, average search time is 30 minutes),
- *do not receive unwanted results* (When using the current search methods, users receive a large collection of links to songs or unrelated sites. Many of these songs do not fit the user taste or preference, and thus are not candidate for a download.),
- *possibility to set multi-choice search criteria* (If users know what they are looking for or need their search results quickly, they want to have search on keywords and to set several search criteria. In other words, users want to customise their search.).
    - search by genre
    - search by content (phrase) or refrain from the song
    - search by year
    - set criteria free or not to download
    - set criteria quality of compression
- *provide manual browsing* (users want to keep manual browsing, because it gives them the possibility to find something new during the search),

3. Concept Description

After studying the agent technology [3,4,5,6] from one hand and the user requirements from another, the following concept was developed.
Today, many users already have some music files on their PCs. If users join their small music databases, there will be a giant distributed music database all over the world. This database can fulfil any kinds of tests of every user.
The purely distributed architecture of the system, when everybody has equal rights and the architecture does not depend on central server, is the most proper solution for such a database. The question of organisation and co-ordination is an important one. Even a big company can't control and co-ordinate such a big and dynamic database. Most of the evidence points out that in such case agents could be used very well. Moreover, they will not only co-ordinate, update, structure, they will also search through this database to find the song the user wants.

In the concept description, several terms are used, that should be introduced before.

- **User community** –the community of users and agents. The users will first – share the music files, second - provide resources for agents' execution. That are the why reasons we call it the user community.
- **Ant** – is the metaphor for an AgeNT. With our agents, we associate hard-working, division of labour, one agent can't do anything, but many of them can do a lot. The same associations can be drawn from ants.
- **Music Information Database (MID)** – a site, which contains information about the artists in the world. The information is the following: artist's albums, songs

from these albums, length of the songs, etc. It is only the textual information. There are no songs, no links to the location of songs on this site.

- **Subnet**: the whole Internet can be divided into subnets. The division can be based on genre: jazz subnet, classical subnet, or on physical location: Eindhoven subnet, Paris subnet. Members of the user community can be in the same subnet or in different subnets. For now, using subnets looks as the most promising alternative for a higher system performance. Subnets allow us to implement mechanisms of "mirror images" (when content of one element of subnet has copies on other elements in subnet). Actually, the idea is that everybody "knows each other very well" in a subnet.

It has been said already that we called the "user community" only because, the users agree to share their music files with other members of the community and users agree to provide some resources of their computer for the agents' execution of other users.

Our system consists of subnets, the users in these subnets, with their music databases, MID and ants, which coordinate the system, search and download songs (see Figure1).

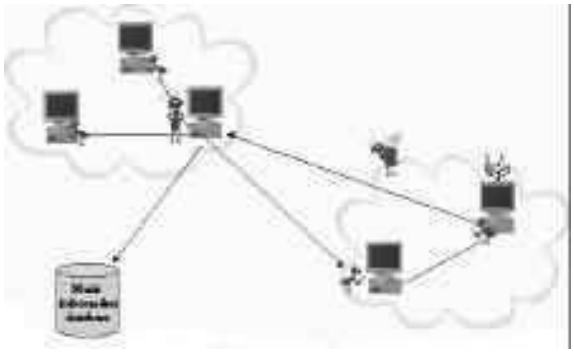


Fig. 1. Search and download

We distinguish several *main steps* in system performance: assign task, search and download.

Assign task. The user needs to type the song or album title or the main artist name. Then, the user's computer connects to the MID to identify the desired song. In order to allow misspelled words or to help users with imperfect memories for titles and names, the user is provided with a list of possible matches, amongst he/she can choose (see Figure 2.).

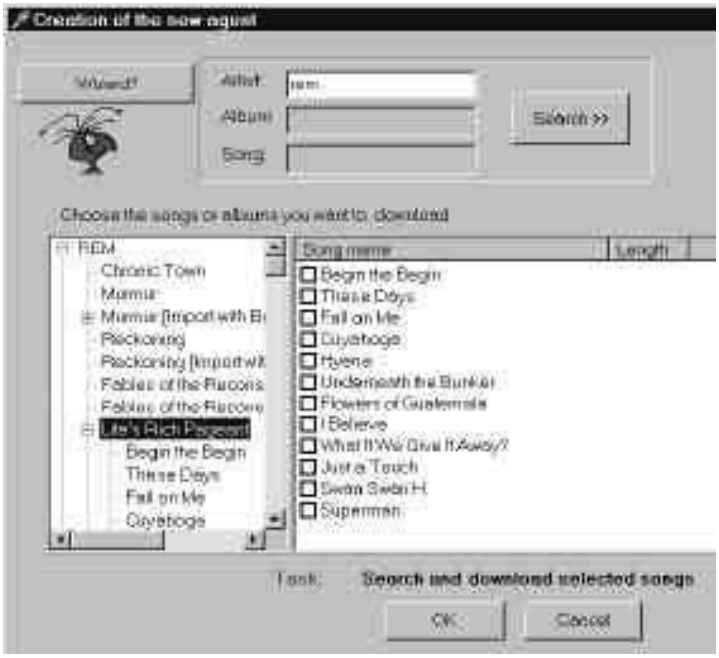


Fig. 2. Information from MID

When the user has chosen the song, he/she can send an ant to look for it and download it. From this step, the user does not directly involve in the search procedure. However, he/she can see the status of an ant, whether the ant is searching whether or not the download is already finished and whether or not the agent has destroyed itself. From the moment the task is set, the user can continue with daily work or even disconnect the computer.

Search. As soon as the ant has the correct name of the song, it goes through the music databases of the other members of the community. There are many possible alternatives what should be the strategies of search. For now, the most productive approach is the subnet-based search, with agent cloning. First, the ant looks for the song within its subnet. If the song cannot be found, the agent searches further on another subnet (see Figure 1.).

Download. In the ideal case, when the user (destination) and the computer (source) are both on-line and the song has been found, the download can start. The ant sets up the connection between the user who wants the song and the user who has the song. When the download has been completed, the ant destroys itself.

When the direct download is not possible, the ant can temp orarily download the song to the ***Virtual Inbox***: a remote mailbox, a friend's computer or remote storage (see Figure 3).

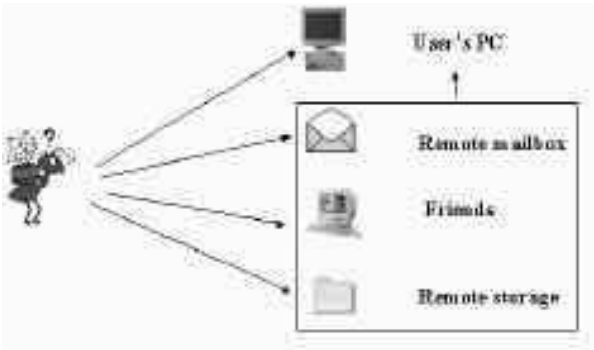


Fig. 3. Virtual Inbox

- **Remote mailbox** – free mail services, like hotmail.
- **Friend's computer** – The user has agreed with a friend, to allow ants downloading songs to the friend's computer when the user is off-line. Obviously, this also holds vice versa. The ant knows the addressees of the friends of the users, so they can be used, when necessary.
- **Remote storage** – the services that are always available on the Internet and provide some space for storage information.

When the user is on-line again, the ant starts immediately download the songs from the temporary storage to the user's computer.

When the song appears on the user's computer, the user is notified by a sound and a flag above a tray icon, for instance.

3.1. Division of Tasks

The main requirement of the mobile agents is to be as small as possible, because they need to travel from one computer to another. In contrast, they need to perform many tasks. We propose to divide tasks among agents (see Figure 4).

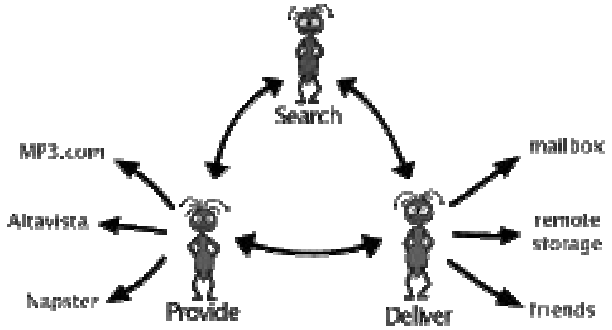


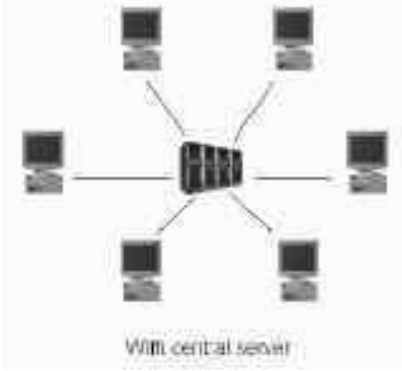
Fig. 4. Division of tasks

Different kinds of agents perform different tasks:

- *Provider agents* know how to access information from different sources, for example mp3.com, altavista.
- *Delivery agents* know particular user information such as how to access a mailbox, how to read from or store information on it, and how to store and access information on the user's friend computer.
- *Search agents* communicate with users and know where to find, *provider* and *delivery* agents.

3.2. Comparison with the Similar Approaches without Agents

Several systems for music search exist already. One of them is Napster [7]. It has the architecture with a central server (see Figure 5). A central server has the information about which songs users have on their PC's and links to them. All connections go through this central server. Information is centralised and search is quicker. The disadvantage of such architecture is that when the central server is shut down, the system does not exist anymore.
Another similar system is Gnutella. It has a purely distributed architecture (Figure 6).



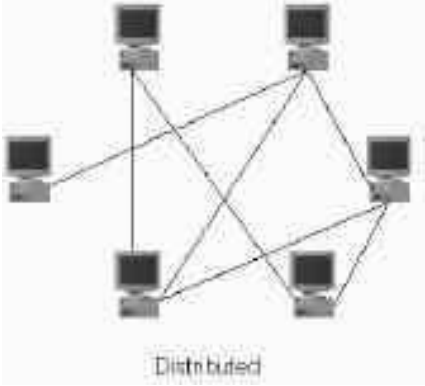


Fig. 5. Central server architecture

Fig. 6. Distributed architecture

The Gnutella protocol is a bit different from the client-server basis in that clients become servers and servers become clients all at once. The playing field is levelled and anyone can be a client or a server, no matter how big or fast they are [8]. The TCP-based Gnutella protocol consumes high bandwidth of your connection, mostly for user-unrelated tasks: communication between hosts, for instance. Such communication approach does not support any ranking mechanism. The system is overloaded quite often spam network traffic.
All these points can be overcome most probably. However, to change the Gnutella protocol, which is a quite complex task considering that the system has been launched already and distributed extensively, software of each client should be updated. That is a main disadvantage of Gnutella: it does not offer a flexible mechanism which can be adapt to the changes in the environment.

In addition to the advantages of our system, which were already described above we will provide the benefits that users gain from music agents:

- save the user time by working autonomously,
- work for the user in his/her absence, even when the computer is turned off,
- learn from interaction with the user or other agents,
- make network resources available to the user.

4. Implementation

We have built a prototype using Java and Voyager ORB. We will describe here its main functionality.

To be a member of the “user community”, user needs to install the “music agent” application on the computer. During the installation, a user gets the start up list o f the addresses of first members of the community. When a new member first starts the application, the Voyager server starts running on the computer.

It is important to keep the list of addresses of other users. Currently, this is solved using the following approach. Every "music agent" application has a list of IP addresses (ipList) of other computers on which the same application has been installed (other members of community). These ipLists are updated each time members are added to or deleted from a community. (This mechanism only holds within a subnet; different subnets communicate using different approach.). So, we assume that all members that are on-line have the same ipList. The first action the agent application should do, when it goes on-line - is to ask any other member, which was already on-line, about the latest, the most recent ipList, and update its own ipList. The second action - is to broadcast the information about itself to all addresses on the ipList. All other members will immediately know about a new member and update their ipLists accordingly. In this way, we solved the problem introducing a new member to the community and of updating ipLusts.

When the user types a keyword (artist name) to search for, the voyager server sends the request to the MID. Till now, it sends the request to allmusic.com. The user receives a list of possible matches to the keyword. The user can browse through the artists’ names, their albums and songs and choose amongst them.

Finally, when the user chose a song, the agent – ant appears. The agent goes to all addresses that are available on the ipList, one-by-one. The agent searches only in directory, which are indicated as searchable.

When the song has been found, the ant sends the request to the voyager server of the user to see whether or not it is possible to download the song. If the server is ready, the ant will send the song. After the server has received the song, it will send a message to the ant that the download has been finished; the ant will destroy itself.

5. Conclusions and Future Work

We used agent technology to improve the way how people search for music files nowadays on the Internet. We have developed a concept according to user requirements, which were acquired in a questionnaire. The main requirements were the following: decrease the time users spend for search, do not receive unwanted

results, manual browsing, and monitor the agents. All these requirements were satisfied.

As the result of the research the conceptual framework of a mobile agent system for music search has been developed.

The concept covers several important aspects:

- user –agent interaction model. This is especially important as long as agent systems are not really understood by most of the people.
- architecture of the agent system. It includes distributed system management, distributed database organization and agent communication mechanisms. The developed system, being developed, is purely distributed, with no central server for organization and communication. Agents, existing on user's computers, solve all questions of communication and organisation without user interaction. When using the system, user should adhere to the following behavioural rule: you want to use the system, you should provide some resources of your computer for agents of other users. Users agree to share their music files and part of their resources with other users, when they are on-line. The agents search on behalf of the users for particular songs in any possible source, including the other computers. System is being developed as self-organised.
- music classification database. Music classification is an important part of music search (but not of an agent system). In our system, it is one or several servers, containing music classification information.

We think that future research might profit from the following directions:

- Search for music meta-data
- Workout a music recommendation function
- Send agents through the firewall
- Development of mechanisms for exchange "if you will give this song, I will give you that song"
- Development of payment scheme
- Distributed system programming and algorithms
- Music classification algorithms.

References

1. D.B. Lange and M. Oshima, Programming and Deploying Java Mobile Agents with Aglets, Addison-Wesley Publishing Co., 1998.
2. Sanchez, J. A, Azevedo, F. S., Leggett, J. J., "PARAgente: Exploring the Issues in Agent-Based User Interfaces", Dept. of Comp Sci, Texas A&M University
3. Gray, R., "Agent Tcl: A flexible and secure mobile-agent system", PhD thesis, Dept. of Comp Sci, Dartmouth
4. Hamilton, G., Cattell, R., Maydene, F., "JDBC Database Access with Java", Addison Wesley, 1997, ISBN 0-201-30995-5.
5. Green, S. et al: Software Agents: A review, IAG Technical Report, Trinity College, May 1997
6. Harrison, C., Chess, D. and Kershenbaum, A.: Mobile agents: Are they a good idea? IBM Research Report, IBM T.J. Watson Research Centre.
7. http://www.napster.com/
8. http://gnutella.wego.com/

Use of Mobile Agents for IPR Management and Negotiation

Isabel Gallego[1, 2], Jaime Delgado[1], Roberto García[1]

[1] Universitat Pompeu Fabra (UPF), Departament de Tecnologia,
La Rambla 30-32, E-08002 Barcelona, Spain
{isabel.gallego,jaime.delgado,roberto.garcia}@tecn.upf.es
[2] Universitat Politècnica de Catalunya, Departament d'Arquitectura de Computadors,
Campus Nord, Mòdul D6, Jordi Girona 1-3, E-08034 Barcelona, Spain
isabel@ac.upc.es

Abstract. The paper describes a model and an implementation of a system to negotiate and manage Intellectual Property Rights (IPR) for the electronic commerce of multimedia products such as video sequences, photos, images, music, documents, etc. This could also be used in the context of electronic publishing services like electronic production of books or multimedia CD-ROMs. The system considers issues from the IPR negotiation phase when a copyrighted material is sold, till the control of later payments derived by rights use. The rights use defines the capabilities that an entity has in relation to authorised operations with copyrighted products. In order to represent the IPR information, metadata are used. In particular, our implementation adapts existing models for a broker based architecture. Finally, the mobile agents are used to allow brokers to collaborate in the analysis and detection of illegal use of copyrighted material.[1]

1 IPR General Model

The starting point, from an IPR's point of view, is the selection of the model in which to base IPR representation and management. The IMPRIMATUR Business Model [1], the one we have selected, identifies a series of entities that may take different roles, such as Creator, Provider, Rights Holder, Distributor, IPR DataBase, or Watermarking & Fingerprint marking.

Figure 1 illustrates the different entities participating in the general model for IPR handling. It must be noted that one user may take more than one different role.

A more simplified and specific model, the one we are producing, consist on the use of a Broker (with the role of Distributor) in charge of being an intermediary between providers of multimedia material (content providers) and the customers interested in buying that material and the corresponding rights for use and/or commercial exploitation. From a functional point of view, these copyrighted multimedia material providers may also assume the roles of Creator and Rights Holders in the same entity.

[1] This work has been partly supported by the Spanish government (TEL98-0699-C02-01).

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 205-213, 2000.

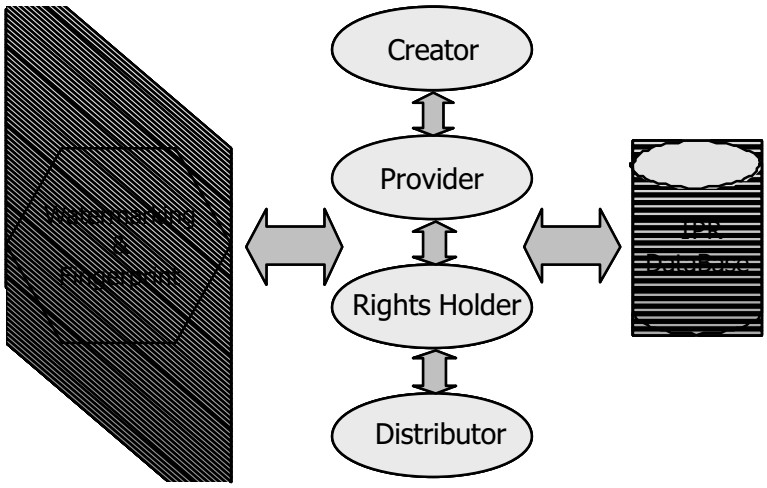


Fig. 1. Different entities in an IPR General Model

Furthermore, the broker stores and keeps up to date (with the help of content providers) the information about the multimedia material for sale in the system (from all content providers associated to the broker), and about the terms and conditions in which commercial electronic transactions are done, with the help of the IPR DataBase. Figure 2 illustrates this Broker Based IPR General Model.

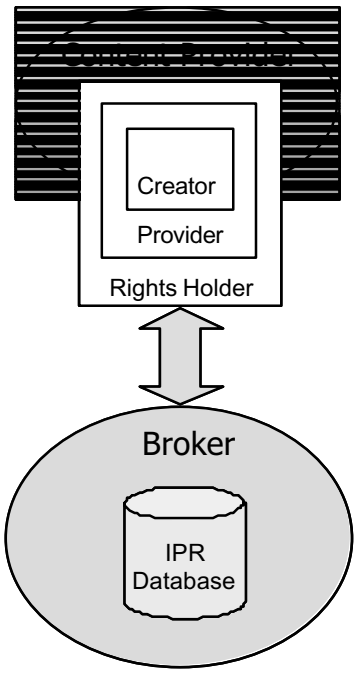


Fig. 2. Broker Based IPR General Model

2 Metadata for Multimedia Content

The model presented in section 1 has been implemented with RDF [2] schemas, extending the fundamental modelling concepts provided by the "RDF model and

syntax" [3] and the "RDF schema" [4]. The schema utilisation policy of this framework allows the creation of a very flexible representation, untied to any concrete initiative of metadata creation.

As said before, our system is based on a broker agent that is trusted by the content providers and the buyers, normally business users. It includes a database with references to the audiovisual material to sell. Since our implementation is focussing on video material, the multimedia metadata (information about the content to sell) stored in the broker can be divided in two types according to its use. First, there are video descriptive attributes (descriptive metadata) used to facilitate that the user locates the audiovisual material he's interested in. The second type encloses the part of the model concerned with intellectual property rights representation (IPR metadata). This information is structured following a metadata model that we have developed based on existing ones, such as those developed by INDECS [5], MPEG-7 [6], CEN/ISSS [7], Dublin Core [8], etc.

To start developing from a solid basis, the model has been constructed over INDECS basic model. This provides the general concepts that allow non-traumatic extensibility and a great adaptability to concrete and changing environments.

Figure 3 shows a hierarchical view of the basic metadata model.

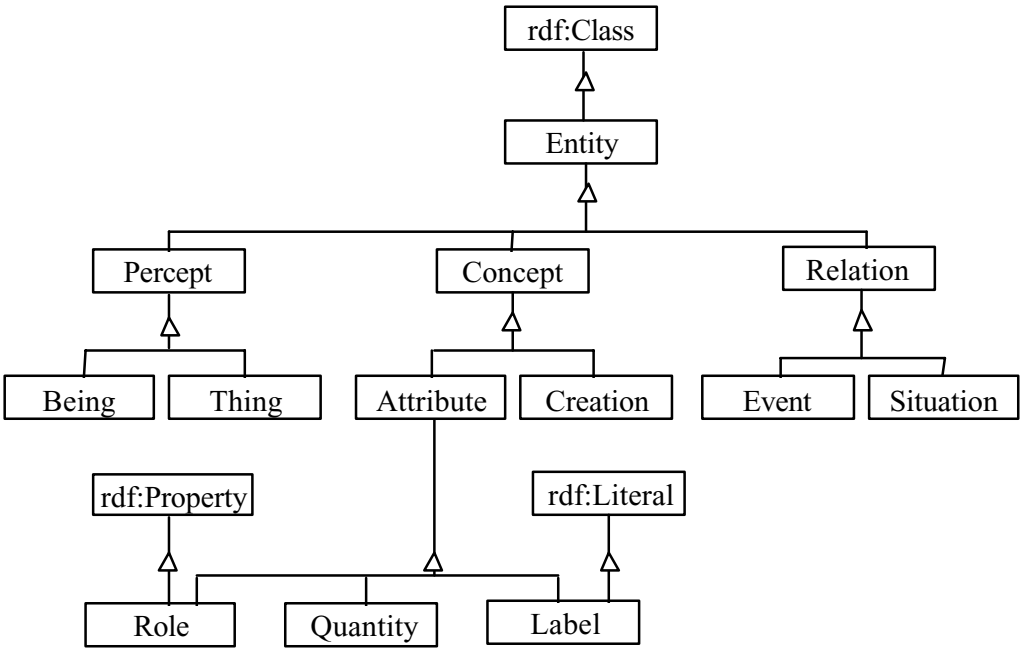


Fig. 3. Basic Metadata Model

It is worth noting that the class "Creation" encloses all the instances of the managed audiovisual material. All these instances could be more strictly classified by defining more specific subclasses, by instance "Video", "Audio", "Picture", …

The descriptive metadata subset includes all those audiovisual material properties that permit the easy and rapid localisation of the audiovisual material needed. These properties can be included according to punctual necessities packaged inside RDF schemes.

Examples of available or definable schemes are the following:

– Dublin Core [8], for general categorisation facilities,

- MPEG-7 [6], for video specific properties, definable with RDF, and
- HYPERMEDIA [9], that uses some video attributes portable to a RDF schema.

All these attributes can be easily included and used to describe the stored audiovisual material. Depending on the semantic applicability of these attributes, they can be associated to the more general representation by the class "Creation" or can only be available to more specific subclasses, like "Video".

The metadata for IPR representation will be detailed in section 4, together with the metadata for IPR negotiation.

3 IPR Negotiation

Based on the IPR attributes set, when a buyer requests, to the broker, a purchase of audiovisual material subject to copyright, the broker extracts IPR information from its database and presents an offer to the buyer. This information allows the buyer to take a decision on how to buy IPR, i.e., to know what are the copyright rules associated to the asset, to decide if he wants to re-sell it, etc. To facilitate this process, a negotiation mechanism is being developed based on a simple negotiation protocol using XML [10]. At first, we have only considered three phases in this simple negotiation protocol: Offer, Agreement and Payment. XML is also used as interface to the IPR information in the broker's database. Since all metadata attributes are specified using XML, the implementation could be easily ported from one system to another. If the negotiation process finishes with agreement, it is complemented by the production of an electronic contract, that is signed by buyer and rights owner and stored in the broker agent.

In an electronic contract, it is necessary to store information about identification of the concerned parties, description of the product, fees, terms of payment, royalties to the author, rights expiry, use rights (re-selling, sub-licensing), etc. Security mechanisms, like digital signature, can be used if we may expect a conflictive situation to appear between the entities who subscribe the contract. The broker agent has, in this sense, a notary role.

If we want to add a signature to the electronic contract, we need to make sure that the electronic contract representation is, when obtained from the information stored in the database, unique. This is necessary to verify the signature and be sure that is a valid one. This problem has not obvious solution depending on how we store the information coming from the RDF representation of the metadata. More details are given in the next section.

4 Metadata for IPR Representation and Negotiation

To represent all the information needed for IPR representation and negotiation, the intellectual property section of the INDECS initiative has been adopted and adapted. This has been constructed over the previous basic metadata model and also takes profit of the RDF facilities.

Figure 4 shows a specific refinement for IPR of part of the hierarchical view already shown in figure 3.

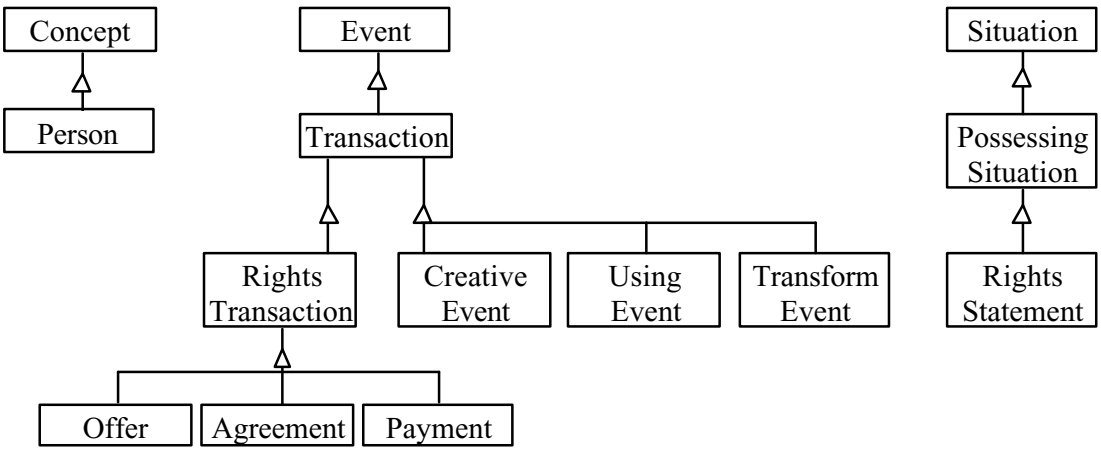


Fig. 4. Metadata for IPR Representation and Negotiation

Two main branches can be observed in the figure, that below the class "Event" model the dynamic aspects treated in IPR negotiation, while the other, enclosed by "Situation", represents the static aspects.

In the dynamic part, IPR commerce is realised by "Rights Transactions", that reference actions over the intellectual property managed, classified in three types of events. These actions and transactions produce determined rights states modelled by "Rights Statements".

Finally, a class has been included to represent all the legal persons concerned in IPR negotiation and management. Under "Person", the entities introduced in the IPR General Model can be included as new subclasses: "Creator", "Provider", "Rights Holder" and "Distributor".

The example of figure 5 represents an agreement between a Rights Holder (content provider) and a Distributor (broker). The notation is based on that used in the INDECS project [5].

As previously introduced, agreements are used as input to the production of electronic contracts. To accomplish this, they should be extracted from the IPR database in a restricted way that allows a unique representation. A first restriction step imposes a depth first search of the agreement graph of figure 5, starting at the "Agreement" class instance, following property edges in alphabetical order. This step will produce a unique sequence of triplets (Subject, Property, Object), the basic conceptual representation of a RDF model.

Now, this sequence is serialised to a XML stream that is lastly signed to produce the required electronic contract. Then, a second restriction step is needed because RDF provides several ways of serialisation of its conceptual model to XML. To avoid ambiguities, the basic abbreviated syntax as specified in RDFMS [3] is imposed as the valid method to produce the XML serialisation of an agreement.

So, starting with the same modelled agreement, these two restriction steps allow the production of unambiguous serialisations that can be digitally signed to produce electronic contracts.

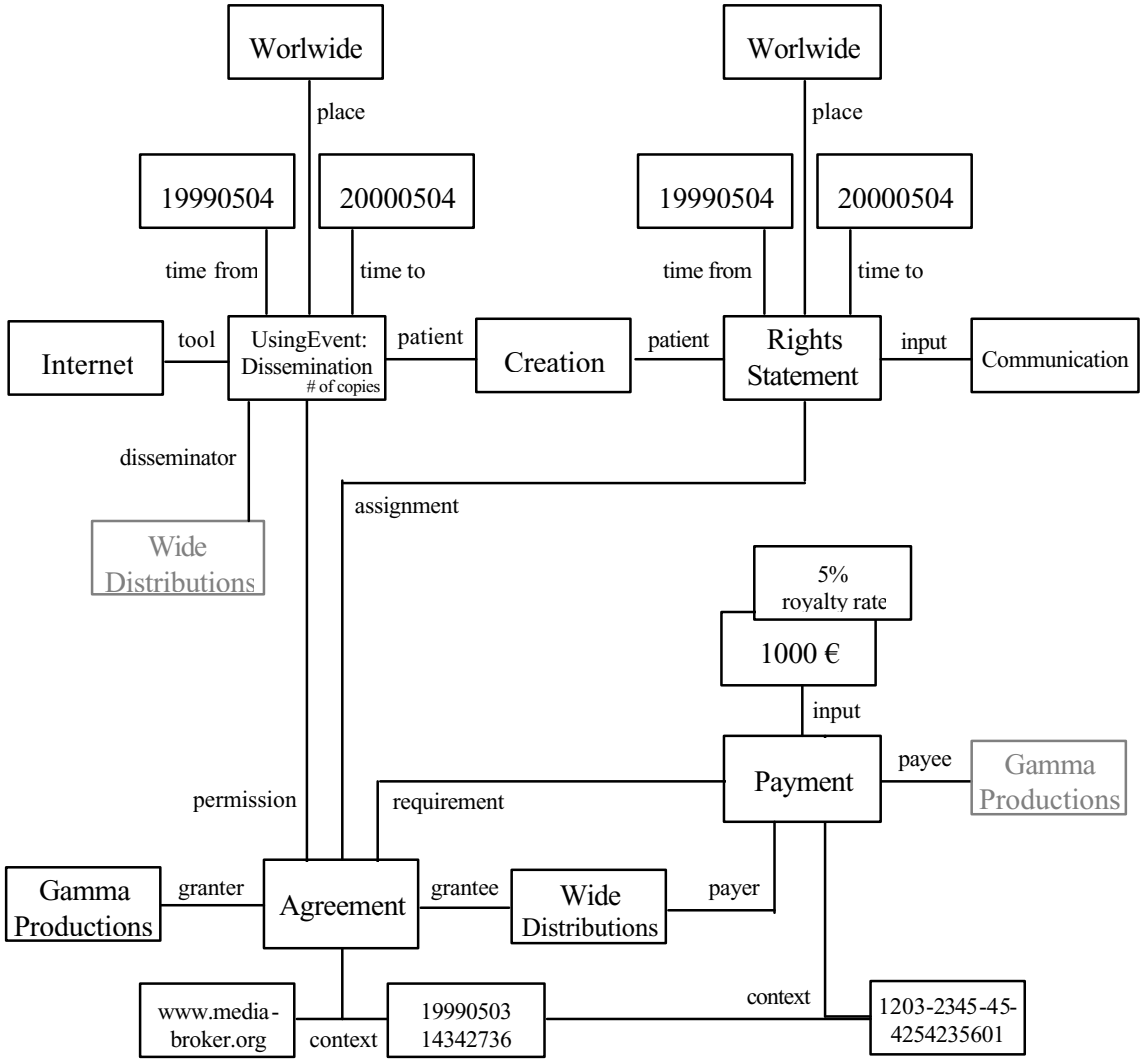


Fig. 5. Schematic example of an Agreement

5 IPR Distributed Management

5.1 Problem and Approach

Different models exist to characterise and formalise electronic commerce. They are based on different points of view, such as the functional, the operational and the architectural one [11]. The operational one specifies the steps followed by a purchase process. The steps are grouped (from a service point of view) in the following four phases: Service Identification, Service Request, Agreement and Post-Agreement. The IPR negotiation process described above finalises in the Agreement phase. However,

in this kind of application it is very important to verify the fulfilment of the contract clauses, normally in a Post-Agreement phase.

There are several issues associated to this phase in relation to IPR management. First, getting payments for the use or re-selling of a copyrighted material. This could be controlled in a "voluntary" basis. Furthermore, if the original buyer informs the broker about the new buyers of the material, then this payment could be easily managed. In this case the broker would have the role of IPR payments collector.

The situation is more complex if the system has the responsibility to follow the fulfilment of the rights associated to products sold without acknowledge information by buyers. In this case, mobile agents can help to control what is happening with the copyrighted material sold by the system. For this, different strategies could be followed. First, checking the WWW to see if audiovisual material without the proper IPR control is circulating. For this purpose, ad-hoc agents might be instructed on how to look for material to check, and how to contact the broker to decide if the situation of that material is legal, i.e., all necessary payments have been done.

A second strategy could be to check if some specific material (a video clip, a video movie, a book, etc.) is in the network without all the requisites. The job for these agents would be to look for material initially bought by a specific user, or look for specific content independently of to whom it has been sold.

5.2 An Implementation with Brokers

For our first implementation, we are assuming an architecture in which the intellectual property rights owners (content providers) are associated to a broker, that is in charge of exploiting (selling) their content rights, and, once those are sold, of controlling that the IPR are respected; i.e., no illegal copies are circulating on the Internet. Figure 6 summarises the architecture.

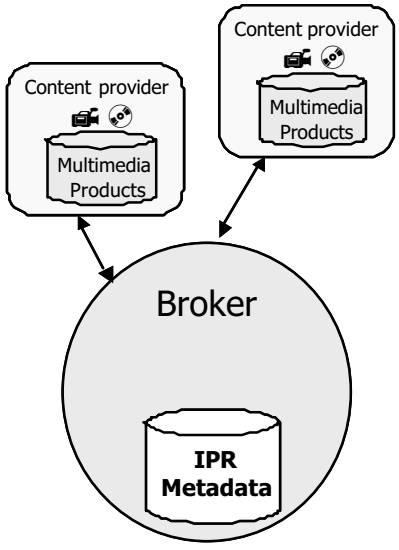


Fig. 6. Broker and Content Providers Architecture

To satisfy the previous requirements, an specific broker includes an specialised service for the distributed management of the IPR of a sold material (what we call IPR Distributed Management Service, or IPR-DMS). This is basically done by searching in the Internet for illegal copies of copyrighted audiovisual material, that is owned by content providers associated to brokers that make use of this IPR-DMS.

As introduced in previous sections, every broker has a database with information (including the IPR related one) about the multimedia material of its associated content providers. This is, for security reasons, private information that should not be transmitted. For this reason, mobile agents are used to facilitate the collaboration between the broker providing the IPR-DMS and the brokers that are making use of this service to protect the IPR of their content providers.

The IPR Distributed Management Service basically offers two different services:

- Continuous search in the Internet.
- Looking for evidences when an illegal situation is detected.

The first service simply consists in accessing WWW servers in the Internet, and verifying the validity of the copies that are publicly available, mainly focussing in material that might be copyrighted by the associated content providers.

The broker providing this service sends a mobile agent to other brokers to verify if the audiovisual material found in an illegal status belongs to them. The mobile agent accesses to their databases and compares their content with the information obtained from the network.

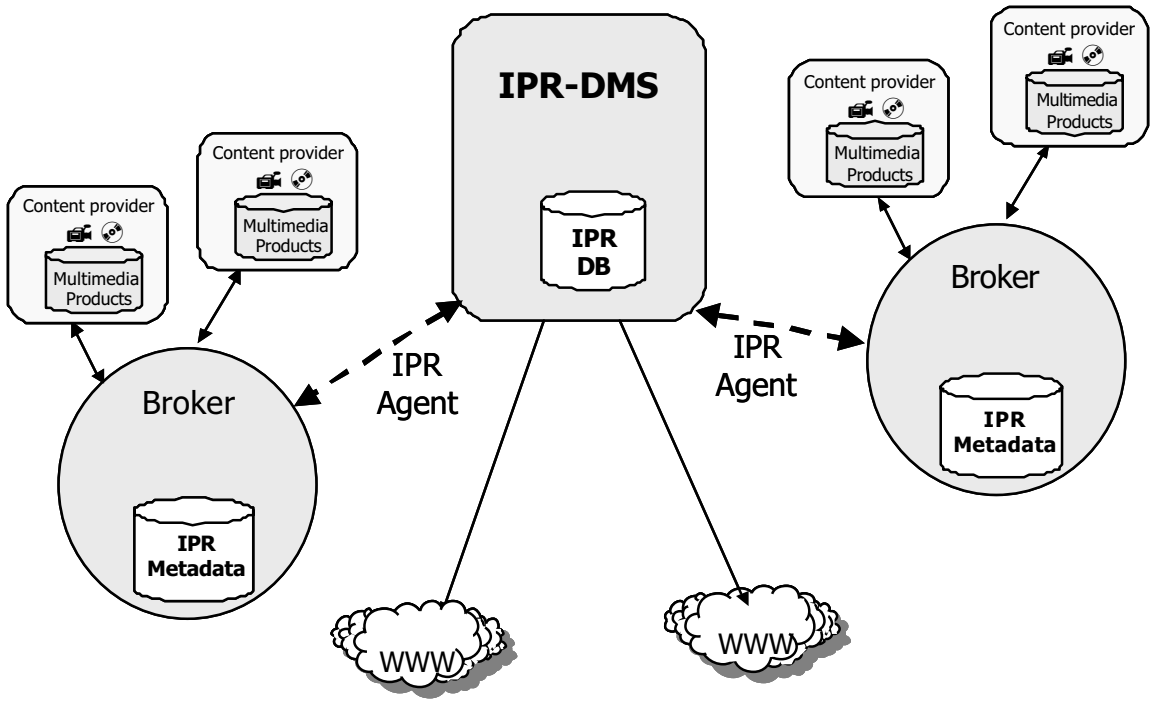


Fig. 7. IPR Distributed Management Architecture

The second service applies when a content provider or a broker suspects that there are illegal copies of its own material on the Internet. In this case, the IPR-DMS

directly looks for the evidences itself, also sending the mobile agent to the relevant broker in order to clarify the evidences.

Figure 7 shows the architecture of the IPR distributed management system, where the different entities are present: IPR-DMS, Brokers and their Content Providers.

6 Conclusions

The paper has shown a specific architecture for management and negotiation of Intellectual Property Rights of audiovisual material in an electronic commerce environment where brokers are used.

These brokers represent content providers and keep metadata about their content, focussing on the IPR related information.

The approach that has been taken with respect to metadata, with a differentiation between descriptive and IPR specific metadata has also been introduced.

The second part of the paper describes the use of mobile agents for distributed management of IPR, focussing on the checking of illegal copies in the Internet.

With mobile agents we facilitate the collaboration between specialised brokers (who provide the IPR Distributed Management Service, IPR-DMS) in order to protect IPR owners.

The representation and negotiation part of our system is already a prototype, while the mobile agents part is still on its specification phase. Further development might show that different approaches could be taken.

References

1. IMPRIMATUR Project, http://www.imprimatur.net
2. RDF (Resource Description Framework), http://www.w3.org/RDF
3. RDFMS (RDF Model and Syntax Specification), http://www.w3.org/TR/REC-rdf-syntax
4. RDFS (RDF Schema Specification), http://www.w3.org/TR/rdf-schema
5. INDECS Project, http://www.indecs.org
6. MPEG-7, http://www.cselt.it/mpeg
7. Metadata for Multimedia Information Workshop, CEN Information Society Standardisation System, http://www.cenorm.be/isss/Workshop/delivered-ws/MMI/Default.htm
8. DUBLIN CORE Home Page, http://purl.oclc.org/metadata/dublin_core
9. HYPERMEDIA Project, http://www.corporacionmultimedia.es/hypermedia
10. XML (eXtensible Markup Language), http://www.w3.org/XML
11. GALLEGO I., DELGADO J. and ACEBRÓN J.J. “Distributed Models for Brokerage on Electronic Commerce”, International IFIP Working Conference Trends in Electronic Commerce (TREC’98), 3-5 June 1998, Hamburg (Germany). Springer, ISBN 3-540-64564-0.

The Effects of Mobile Agent Performance on Mp3 Streaming Applications

B. Thai, A. Seneviratne

binh@ee.unsw.edu.au, a.seneviratne@unsw.edu.au
School of Electrical Engineering and Telecommunications
University of New South Wales, Sydney, Australia

Extended Abstract

There are many areas in which mobile agent technology can be deployed. Researchers are now looking at this technology as a solution in their respective research areas, such as information retrieval, distributed computing, and network management. This increase of interest is due the basic characteristics of mobileagents – autonomous and mobile. In the paper, we are exploiting the abovementioned characteristics of mobile agents, and applying this technology in Internetenvironments. Using a mobile agent as a transparent content transcoding proxy in a proxy based network architecture, we show how a mobile proxy agent can be used to improve quality of service provided to the end user connected via heterogeneous access networks, using computing devices which have very different capabilities.

In our framework, we are using software agents, both static and mobile, programmed with different proxy functions, to provide transparent content transcoding and network enhancements to the end user. This is an extension to the current proxy-based network architectures [1], which uses dedicated and centralised proxy servers to provide the transparent transcoding. Using mobile agents make our framework more efficient, flexible and scalable than the static proxy architectures, as a proxy agent can follow a roaming user, execute on a lightly loaded server, and/or optimally located to keep the number of hops between itself and the end user to a minimum.

There have been different performance measurements on various mobile agent platforms available today. Narasimhan [3, 4, 5] performed two experiments on IBM aglets to determine the effect of network congestion and the size of the aglet affects migration time. The first experiment was measuring the round-trip time of an aglet with increasing size, and the second experiment was measuring the round-trip time of an aglet when it travels around the world via various hosts. The results provide us with a rough estimate of the performance of the IBM aglets; however, due to the simplicity of the experiments, we cannot draw any conclusions on the suitability of aglets for our framework from this work. Silva et al. [7] extensively benchmarked 8 different mobile agent platforms: IBM Aglets, Concordia, Voyager, Odyssey, Jumping Beans, Grasshopper, Swarm and James. They performed 12 experiments, testing each mobile agent platform on execution time, migration time, the amount of network traffic and robust-

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 215-217, 2000.

ness. They concluded that James provides the best performance out of the 8 mobile agent platforms they evaluated. We can use this study as a guideline on which mobile agent platform has the best performance, but we cannot use this study to decide the viability of the use of mobile agents in our framework, because the experiments Silva et al. performed used unrealistic scenarios.

Moreover, the study of Narasimhan and Silva et al., were aimed at measuring the performance of mobile agents, as a result they were not consider the specific application of the technology. Therefore the results they presented are only the performance of the mobile agent in a general case. To evaluate the viability of our mobile agent based proxy architecture, we developed a prototype mp3 audio streaming application using IBM aglets. Our experiment concentrated on the migration time of the aglets, and the perceived quality of the MP3 stream without any transcoding, and with a mobile aglet transcoder. The prototype used a modified version of Obsequieum [6] as a streaming mp3 server, and Freeamp [2] as a streaming mp3 client. Unlike other streaming mp3 applications, which use TCP as their streaming protocol, Obsequieum streams mp3 audio using RTP.

To test the performance of the system we conducted two experiments. The aim of the first experiment was to determine the amount of audio lost when the proxy agent migrates from one host to another. The results we obtained from the experiment agreed with our previous streaming audio emulation results, that the number of audio packet loss is directly related to the network traffic. However, since RTP contains time stamping, we discovered that on average, the minimum amount of audio loss is 240ms. Under heavy network traffic, the amount of audio loss can be as much as 1700ms. We also discovered the effect of the client application with buffering and without buffering. When buffering is on (we used the default value of 3 seconds), at minimum network traffic, the user only hears a slight skip of audio – the 240ms loss of audio. However, as the buffer empties due to the high frequency of migration, the application stops playing the audio for 3 seconds for re-buffering. From the end user's perspective, s/he experiences silence for 3 seconds and still suffer a 240ms loss of audio. The situation is more severe under heavy network traffic. As the application buffer appears to be emptied at every agent migration, the application has to re-buffer. This causes a period of silence for 3 seconds and a loss of 1700ms of audio. The second experiment was aimed at investigating how a proxy agent can minimise the loss of audio.

A test network is set up, which consists of 3 different subnets. The mobile client can roam between subnet B and C, and the proxy agent can follow the mobile client by residing on base station B and C. Subnet B is a congested network. The experiment involved the mobile client roaming between subnet B and C while the server is streaming mp3 audio via a mobile proxy agent. We are hypothesising that if subnet B congested, the amount of audio loss will be high. If the mobile client roams to subnet C, but the mobile proxy agent remains in base station B, the amount of audio loss will be higher, since the RTP packets has to go through the congested network twice. However, if the mobile proxy agent migrates to base station C, there will be a period of which there will not no audio, but once the migration process completes, the amount of audio loss should be less than before, as the audio packets are no longer route

through subnet B. The initial results show that the above assertion is valid, and that the overall perceived quality is improved.

References

[1] Fox A, Gribble S, Chawathe Y and Gribble E, "Adapting to Network and Client Variations using Infrastructional Proxies: Lessons and Perspective", IEEE Personal Communications, August , 1998

[2] Freeamp homepage, http://www.freeamp.org

[3] Narasinhan N, "Experiments To Evaluate Aglet Latency", http://www.beta.ece.ucsb.edu/~nita/agletsExpt/index.html

[4] Narasinhan N, "Latency for Aglet Travel Around The World", http://www.beta.ece.ucsb.edu/~nita/agletExpt/agletsExpt2.html

[5] Narasinhan N, "Variation in Aglet Latency with Aglet Size", http://www.beta.ece.ucsb.edu/~nita/agletExpt/agletsExpt1.html

[6] Obsequieum hoempage, http://obs.freeeamp.org

[7] Silva L M et al, "Comparing the performance of mobile agent system: a study of bench-marking", Elsevier Computer Communications, Issue 23, p.769-778.

Semi-trusted Hosts and Mobile Agents: Enabling Secure Distributed Computations*

Bart De Decker, Frank Piessens**, Erik Van Hoeymissen, Gregory Neven

Dept. of Computer Science, K.U.Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium
Fax: ++32(0)16 327996
E-mail:{bart,frank,erikv,gneven}@cs.kuleuven.ac.be

Abstract. Secure distributed computing addresses the problem of performing a computation with a number of mutually distrustful participants, in such a way that each of the participants has only limited access to the information needed for doing the computation. In the presence of a third party, completely trusted by all participants the problem is trivially solvable. However, this assumption is in many applications non-realistic. Over the past two decades, a number of solutions requiring no trusted third party have been developed using cryptographic techniques. The disadvantage of these cryptographic solutions is the excessive communication overhead they incur.
In this paper, we will show how to overcome these disadvantages and thus enable secure distributed computations in practice. Our approach uses mobile agents employing these cryptographic techniques to provide for a trade-off between communication overhead and trust. The communication overhead problem would be solved if the communicating parties were brought close enough together. Our solution is to use mobile agents to execute the cryptographic protocols. Of course, a mobile agent needs to trust his execution platform, but we show that the trust requirements in this case are much lower than for a classical trusted third party.

1 Introduction

Secure distributed computing (SDC) addresses the problem of distributed computing where some of the algorithms and data that are used in the computation must remain private. Usually, the problem is stated as follows, emphasizing privacy of data. Let f be a publicly known function taking n inputs, and suppose there are n parties (named $p_i, i = 1 \ldots n$), each holding one private input x_i. The n parties want to compute the value $f(x_1, \ldots, x_n)$ without leaking any information about their private inputs (except of course the information about x_i that is implicitly present in the function result) to the other parties. An example is voting: the function f is addition, and the private inputs represent yes ($x_i = 1$)

* A preliminary version of this paper was presented at the ECOOP 2000 Workshop on Mobile Object Systems

** Postdoctoral Fellow of the Belgian National Fund for Scientific Research (F.W.O.)

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 219-230, 2000.

or no ($x_i = 0$) votes. In case you want to keep an algorithm private, instead of just data, you can make f an interpreter for some (simple) programming language, and you let one of the x_i be an encoding of a program.

In descriptions of solutions to the secure distributed computing problem, the function f is usually encoded as a boolean circuit, and therefore secure distributed computing is also often referred to as *secure circuit evaluation.*

It is easy to see that an efficient solution to the secure distributed computing problem would be an enabling technology for a large number of interesting distributed applications across the Internet. Some example applications are: auctions ([13]), charging for the use of algorithms on the basis of a usage count ([14, 15]), various kinds of weighted voting, protecting mobile code integrity and privacy ([15, 11]), ...

Secure distributed computing is trivial in the presence of a globally trusted third party(TTP): all participants send their data and code to the TTP (over a secure channel), the TTP performs the computation and broadcasts the results. The main drawback of this approach is the large amount of trust needed in the TTP.

However, solutions without a TTP are also possible. Over the past two decades, a fairly large variety of solutions to the problem has been proposed. An overview is given by Franklin [7] and more recently by Cramer [5]. These solutions differ from each other in the cryptographic primitives that are used, and in the class of computations that can be performed (some of the solutions only allow for specific kinds of functions to be computed). The main drawback of these solutions is the heavy communication overhead that they incur. For a case-study investigating the communication overhead in a concrete example application, we refer the reader to [12].

Mobile agents employing these cryptographic techniques can provide for a trade-off between communication overhead and trust. The communication overhead is alleviated if the communicating parties are brought close enough together. In our approach, every participant sends its representative agent to a trusted execution site. The agent contains a copy of the private data x_i and is capable of running a SDC-protocol. Different participants may send their agents to different sites, as long as these sites are located closely to each other. Of course, a mobile agent needs to trust his execution platform, but we show that the trust requirements in this case are much lower than for a classical TTP. Also, in contrast with protocols that use unconditionally TTPs, the trusted site is not involved directly. It simply offers a secure execution platform: i.e. it executes the mobile code correctly, does not spy on it and does not leak information to other mobile agents. Moreover, the trusted host does not have to know the protocol used between the agents. In other words, the combination of mobile agent technology and secure distributed computing protocols makes it possible to use a *generic* TTP that, by offering a secure execution platform, can act as TTP for a wide variety of protocols in a uniform way. A detailed discussion of the use of mobile agent technology for advanced cryptographic protocols is given in section 3.

The combination of cryptographic techniques for secure computing and mobile code has been investigated from another point of view by Sander and Tschudin ([14, 15]). In their paper on mobile cryptography, they deal with the protection of mobile agents from possibly malicious hosts. Hence, the focus in their work is on the use of cryptographic techniques for securing mobile code. The security concerns posed by the mobile agent protection problem are code privacy (Can a mobile agent conceal the program it wants to have executed?), code and execution integrity (Can a mobile agent protect itself against tampering by a malicious host?) and computing with secrets in public (Can a mobile agent remotely sign a document without disclosing the user's private key?). To address some of these concerns, cryptographic secure computation techniques can be used. We discuss this in more detail in section 2.3, which is part of our survey on secure distributed computing protocols.
The structure of this paper is as follows. In the next section, we start of with a survey of existing cryptographic solutions to the secure computing problem. In section 3, we introduce and compare three possible ways to implement secure distributed computing, making use of both cryptographic techniques, and trusted parties. The comparison is done based on a simple model of the trust and communication requirements for each of the solutions. Finally, in section 4, we summarize the main outcome of this comparison.

2 Survey of SDC protocols

Various kinds of solutions for the secure distributed computing problem have been proposed in the literature (often using different terminology than the one used in this paper).

2.1 Using probabilistic encryption

One class of techniques to compute with encrypted data is based on *homomorphic probabilistic encryption*. An encryption technique is *probabilistic* if the same cleartext can encrypt to many different ciphertexts. To work with encrypted bits, probabilistic encryption is essential, otherwise only two ciphertexts (the encryption of a zero and the encryption of a one) would be possible, and cryptanalysis would be fairly simple. An encryption technique is *homomorphic* if it satisfies equations of the form $E(x \quad \mathbf{op} \quad y) = E(x) \quad \mathbf{op}' \quad E(y)$ for some operations **op** and **op**$'$. A homomorphic encryption scheme allows operations to be performed on encrypted data, and hence can be used for secure circuit evaluation.
Abadi and Feigenbaum present a protocol for two-player secure circuit evaluation using a homomorphic probabilistic encryption scheme based on the Quadratic Residuosity Assumption (QRA) in [1]. This protocol allows A who has a secret function f and B who has secret data x to calculate $f(x)$ without revealing their secrets.
Let k be the product of two primes p and q, each congruent to 3 mod 4. An integer $a \in Z_k^*[+1]$ — the integers relatively prime to k with Jacobi symbol 1 — is a quadratic residue mod k if there exists an $x \in Z_k^*[+1]$

such that $a = x^2 \bmod k$. The QRA states that determining if an integer a is a quadratic residue mod k is a hard problem if the factorization of k is unknown but is easy to solve if p and q are given.
If we encrypt a zero by a quadratic residue and a one by a quadratic nonresidue mod k, we can define the encryption of a bit b as

$$E_k(b) = (-1)^b \cdot r^2 \bmod k$$

with $r \in_R Z_k^*[+1]$ chosen at random. This probabilistic encryption scheme has two homomorphic properties that will come in handy in the protocol:

$$E_k(\overline{b}) = (-1) \cdot E_k(b) \bmod k$$

$$E_k(b_1 \oplus b_2) = E_k(b_1) \cdot E_k(b_2) \bmod k$$

B starts the protocol by choosing p and q and multiplying them to produce k. B sends k and the encryption of his data bits $E_k(x_1), \ldots, E_k(x_n)$ to A. B keeps the factorization of k secret. A then starts evaluating her secret circuit. If she has to evaluate a NOT gate with input $E_k(b)$, she simply calculates $-E_k(b) \bmod k$. An XOR with inputs $E_k(b_1)$ and $E_k(b_2)$ is also easy to evaluate: A just takes $E_k(b_1) \cdot E_k(b_2) \bmod k$ as the output of the gate. To evaluate the AND of inputs $E_k(b_1)$ and $E_k(b_2)$, she needs B's help. A chooses two bits c_1 and c_2 at random and sends $E_k(b_1 \oplus c_1)$ and $E_k(b_2 \oplus c_2)$ to B. B decrypts the bits A just sent him as d_1 and d_2 (he can do so because he knows p and q) and sends the tuple

$$< E_k(d_1 \wedge d_2), E_k(d_1 \wedge \overline{d_2}), E_k(\overline{d_1} \wedge d_2), E_k(\overline{d_1} \wedge \overline{d_2}) >$$

to A. A takes the first element of this tuple as the output of the AND gate if she chose $c_1 = c_2 = 0$, the second if she chose $c_1 = 0$ and $c_2 = 1$, the third if she chose $c_1 = 1$ and $c_2 = 0$ and the last one if she chose $c_1 = c_2 = 1$. Proceeding this way from gate to gate, A ends with the encrypted result $E_k(f(x))$ and sends it for decryption to B.
Note the large amount of communication overhead in the protocol: for each AND gate to be evaluated, a large amount of communication is necessary. Concrete estimates of the communication overhead in a realistic example can be found in [12]

2.2 Protocols based on oblivious transfer

In [9], Goldreich, Micali and Wigderson present a two-party protocol for the problem of *combined oblivious transfer* which is equivalent to the problem of secure circuit evaluation. The setting is slightly different than in the previous protocol. Here, two parties A and B want to evaluate a publicly known boolean circuit. This circuit takes input from both A and B, but each party wants to keep his part of the data private. In contrast, in the previous protocol, the circuit was private to A, and the data was private to B. Recall from the introduction that these two settings are essentially equivalent: by making the publicly known circuit a universal circuit, it is still possible to hide functions instead of data.
The basic idea of the protocol we are about to describe is the following: A will evaluate the circuit, not on the actual bits, but on encodings of

those bits. The encoding of the bits is known only to B. So A evaluates the circuit, but can not make sense of intermediary results because she doesn't know the encoding. B knows the encoding but never gets to see the intermediary results. When the final result is announced by A (in encoded form), B will announce a decoding for this final result.
We give a more detailed description of the protocol. B assigns two random bit strings r_i^0 and r_i^1 to every wire i in the circuit, which represent an encoded 0 and 1 on that wire. This defines a mapping $\phi_i : r_i^b \mapsto b$ for every wire i. B also chooses a random bit string R that will allow A to check if a decryption key is correct. The general idea of the protocol is that, if b is the bit on wire i in the evaluation of the circuit for A's and B's secret inputs, A will only find out about r_i^b and will never get any information about $\phi_i(r_i^b)$ or $r_i^{\overline{b}}$. In other words, A evaluates the circuit with encoded data.
We use the notation $E(M, r)$ for a symmetric encryption function of the message M with secret key r. To encrypt a NOT-gate with input wire i and output wire o, B constructs a random permutation of the tuple

$$< E(R \cdot r_o^1, r_i^0), E(R \cdot r_o^0, r_i^1) >$$

where $\cdot$ denotes the concatenation of bit strings. To encrypt an AND-gate with input wires l and r and output wire o, B constructs a random permutation of the tuple

$$\begin{aligned} < E(R \cdot r_o^0, r_l^0 \oplus r_r^0), E(R \cdot r_o^0, r_l^0 \oplus r_r^1), \\ E(R \cdot r_o^0, r_l^1 \oplus r_r^0), E(R \cdot r_o^1, r_l^1 \oplus r_r^1) > \end{aligned}$$

with $\oplus$ the bit-wise XOR. Any other binary port can be encrypted in an analogous way.
B sends the encryption of every gate in the circuit together with R, the encoding of his own input bits and the mapping ϕ_m of the output wire m to A. To perform the evaluation of the circuit on encoded data, A first needs encodings of all the input bits. For B's input bits, the encoding was sent to her, but since B doesn't know A's inputs, B can't send an encoding of them. Note that B can't send the encoding of both a 1 and a 0 on A's input wires either, because that would allow A to find out more than just the result of the circuit. The technique that is used to get the encoding of A's input to A is called *one-out-of-two oblivious transfer* ([6]). This is a protocol that allows A to retrieve one of two data items from B in such a way that (1) A gets exactly the one of two items she chose and (2) B doesn't know which item A has got.
Thus, A and B execute a one-out-of-two oblivious bit string transfer (often referred to as $\binom{2}{1}$-OT^k) for each of A's input bits. This guarantees that A only obtains the encoding of her own input bits without releasing any information about her bits to B. A evaluates each gate by trying to decrypt every element of the tuple using the encoding of the bit on the input wire (or the XOR of two input bit encodings) as a key; she will only decrypt one of the elements successfully, thereby obtaining the encoded bit on the output wire. Note that she can verify if a decryption was correct by comparing the first bits of the decrypted string with R.

Proceeding this way through the entire circuit, A obtains the encoding of the final output and applies ϕ_m to reveal the plain output bit.
Another protocol for two-party secure computation based on oblivious transfer is presented in [10]. The basic idea in this protocol is to have the participants compute the circuit on data that is shared by the two parties using a technique known as *secret sharing*.

2.3 Autonomous protocols

The protocols discussed in the two previous subsections require more communication rounds than strictly necessary. The probabilistic encryption based protocol requires one communication round per AND-gate in the circuit. The oblivious transfer based protocol requires one communication round for performing the oblivious transfer of the input, and another for sending the encrypted circuit.
For protecting mobile code privacy and integrity, non-interactive (or autonomous) protocols are necessary ([15]). The idea here is to realize a system where a host can execute an encrypted function without having to decrypt it. Thus, functions would be encrypted such that the resulting transformation can be implemented as a mobile program that will be executed on a remote host. The executing computer will be able to execute the program's instructions but will not be able to understand the function that the program implements. Having function and execution privacy immediately yields execution integrity: an adversary can not modify a program in a goal-oriented way. Modifying single bits of the encrypted program would disturb its correct execution, but it is very hard to produce a desired outcome.
It turns out to be possible to construct such autonomous solutions where the client sends (in one message) an encrypted function f, and it receives from the server an encrypted result $f(x)$ in such a way that f remains private to the client and x remains private to the server.
Various autonomous protocols have been proposed in the literature. Sander and Tschudin ([14, 15]) introduce a technique that allows for a fairly efficient evaluation of polynomials in a ring of integers modulo n using a homomorphic encryption scheme. They also show how an autonomous protocol could be realized using compositions of rational functions.
Sander and Tschudin emphasize in their paper that securing single functions is not sufficient. Consider for example the problem of implementing a digital signing primitive for mobile agents. Even if the real signature routine can be kept secret, still the whole (encrypted but operational) routine might be abused to sign arbitrary documents. Thus, the second task is to guarantee that cryptographic primitives are unremovably attached to the data to which they are supposed to be applied (the linking problem). The general idea behind the solution here is to compose the signature generating function s with the function f of which the output is to be signed. Crucial for the security of this scheme is the difficulty of an adversary to decompose the final function into its elements s and f. An outline of how this could be implemented using rational functions is given in [15].

Loureiro and Molva ([11]) use a public key encryption system based on Goppa codes. Their protocol allows for the evaluation of functions describable by a matrix multiplication. Loureiro and Molva also show how any boolean circuit evaluation can be done by a matrix multiplication. However, the representation of a boolean circuit requires a *huge* matrix (for a circuit with l inputs, one of the dimensions of the matrix is 2^l). It remains an open problem whether more efficient representations of boolean circuits as matrices can be achieved.
Finally, two very recent papers also focus on autonomous protocols: Sander, Young and Yung ([16]) propose an autonomous protocol based on a new homomorphic encryption scheme, and Cachin, Camenisch, Kilian and Müller ([2]) start from an OT-based SDC protocol as in section 2.2, and succeed in merging the two phases of this protocol into one.
It is worth emphasizing that, even though autonomous protocols use the minimal number of messages, they do not solve the communication overhead problem: even though there are only two messages exchanged, these messages are extremely big.

2.4 Multi-party protocols

All the previous protocols concentrate on the two-party case: only two parties are involved in the secure computation process. It is clear that the multi-party case is even more interesting from an application-oriented point of view. The multi-party case has also received considerable interest in the literature.
Chaum, Damgård and van de Graaf present a multi-party protocol in [3] that starts with the truth table of every gate in the circuit. Each player in turn receives a "scrambled" version of the truth tables from the previous player, transforms the truth tables by adding his own encryptions and permutations, commits to his encryptions and sends these transformed truth tables to the next player. When the last player finished his transformation, all players evaluate the scrambled circuit by selecting the appropriate row from the truth tables.
Franklin and Haber present an elegant multi-party protocol based on group-oriented cryptography in [8]. All parties send each other an ElGamal alike joint encryption of their input bits and evaluate the entire circuit together. The evaluation of a NOT-gate can be done without interaction while the evaluation of an AND-gate requires broadcasting encrypted bits and "decryption witnesses". Finally, each party sends a decryption witness for the output bit.
Even information-theoretically secure multi-party computation can be achieved (as opposed to only computationally secure). A possible realisation is discussed in [5].
The communication overhead for multi-party protocols is even more serious than that for the two-party protocols.

3 Trust versus Communication Overhead

In this section, the different options for implementing secure distributed computation are discussed. It will be shown that there is a trade-off

between trust and communication overhead in secure computations. If all participants are distrustful of each other, the secure computation can be performed using protocols surveyed in the previous section with a prohibitive huge amount of communication. However, if a TTP is involved, the communication overhead can be made minimal.

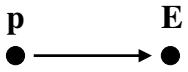
Recall from section 1 that f is a publicly known function taking n inputs. Assume that there are n distrustful participants $p_1, ..., p_n$, each holding one private input x_i. The n participants want to compute the value of $f(x_1, ..., x_n)$ without leaking information of their private inputs to the other participants.

To compare the trust requirements of the different approaches, we use the following simple trust model. We say a participant *trusts* an execution site if it believes that:

- the execution site will correctly execute any code sent to it by the participant;
- the execution site will correctly (i.e. as expected by the participant) handle any data sent to it by the participant.

It also implies that the execution site will maintain the privacy of the data or the code if this is expected by the participant.

If p trusts E, we denote this as:



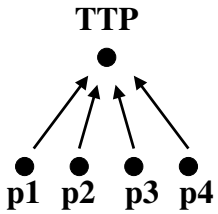
To compare bandwidth requirements (for communication overhead), we make the following simple distinction. *High* bandwidth is required to execute a SDC protocol. *Low* bandwidth suffices to transmit data or agent code. We assume low bandwidth communication is available between any two sites. If high bandwidth communication is possible between E_i and E_j, we denote this as follows:

To see that this simple two-valued model of bandwidth requirements is sufficient for our case, we refer the reader to [12]. In that paper, a case-study investigating the communication overhead for a so-called *Secret Query Database* is given. In this application, A has a query q and B owns a database with records x. The Secret Query Database allows them to cooperate in such a way that they can compute $q(x)$ while A preserves the secrecy of q and B preserves that of x. The communication overhead to solve this concrete case with SDC protocols is in the order of magnitude of 100 megabytes. On the other hand, sending just the query data, or sending an agent containing the query requires only a few kilobytes of communication. The large difference in amount of communication shows that our simplified model of high and low bandwidth requirements is realistic.

Based on these simple models of communication and trust, we compare the three options for implementing secure distributed computations.

3.1 A Trusted Third Party

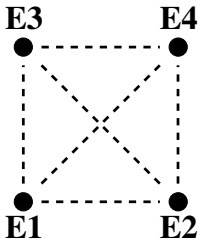
The first, perhaps most straightforward option, is to use a globally trusted third party. Every p_i sends its private input x_i to the TTP who will compute $f(x_1, ..., x_n)$ and disseminate the result to the participants $p_i, i = 1..n$.
Of course, before sending its private data to the TTP, every p_i must first authenticate the TTP, and then send x_i through a safe channel. This can be accomplished via conventional cryptographic techniques.
It is clear that this approach has a very low communication overhead: the data is only sent once to the TTP; later, every participant receives the result of the computation. However, every participant should unconditionally trust the TTP. For the case of 4 participants, the situation is as follows:



It is not clear whether n distrustful participants will easily agree on one single trustworthy execution site. This requirement of one single globally trusted execution site is the main disadvantage of this approach.

3.2 Cryptographic Secure Distributed Computing

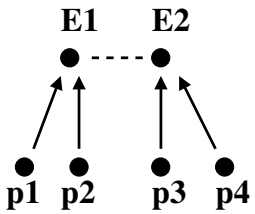
The second option is the use of cryptographic techniques (as surveyed in section 2) that make the use of a TTP superfluous.
The trust requirements are really minimal: every participant p_i trusts its own execution site E_i, and expects that the other participants provide correct values for their own inputs.
Although this option is very attractive, it should be clear from the previous sections and from [12] that the communication overhead is far too high to be practically useful in a general networked environment. High bandwidth is required between all of the participants. For the case of 4 participants, the situation can be summarized as follows:



3.3 A Virtual Trusted Third Party

Finally, our solution tries to combine the two previous options: the communication overhead of SDC-techniques are remedied by introducing semi-trusted execution sites and mobile agents.

In this approach, every participant p_i sends its representative, agent a_i, to a trusted execution site E_j. The agent contains a copy of the private data x_i and is capable of running a SDC-protocol.
It is allowed that that different participants send their agents to different sites. The only restriction being that the sites should be located closely to each other, i.e. should have high bandwidth communication between them.
Of course, every execution site needs a mechanism to safely download an agent. However, that can be easily accomplished through convential cryptographic techniques.
The amount of large distance communication is moderate: every participant sends its agent to a remote site, and receives the result from its agent. The agents use a SDC-protocol, which unfortunately involves a high communication overhead. However, since the agents are executing on sites that are near each other, the overhead of the SDC-protocol is acceptable.
For a situation with 4 participants, we could have the following situation:



No high bandwidth communication between the participants is necessary, and there is no longer a need for one single trusted execution site. p_1 for example, does not need to trust site E_2. The agents that participate in the secure computation are protected against malicious behaviour of other (non-trusted) execution sites by the SDC-protocols. That is sufficient to make this approach work.
Moreover, in contrast with the approach where one uses an unconditionally trusted third party, the trusted sites are not involved directly. They simply offer a secure execution platform: the trusted hosts do *not* have to know the protocol used between the agents. In other words, the combination of mobile agent technology and secure distributed computing protocols makes it possible to use *generic* trusted third parties that, by offering a secure execution platform, can act as trusted third party for a wide variety of protocols in a uniform way.
Finally, the question remains whether it is realistic to assume that participants can find execution sites that are close enough to eachother. Given the fact however that these execution sites can be *generic*, we believe that providing such execution sites could be a commercial occupation. Various deployment strategies are possible. Several service providers, each administering a set of geographically dispersed "secure hosts", can propose their subscribers an appropriate site for the secure computation. The site is chosen to be in the neighbourhood of a secure site of the other service providers involved. Another approach is to have execution parks, offering high bandwidth communication facilities, were companies can install their proprietary "secure site". The park itself could be managed by a commercial or government agency.

4 Conclusion

This paper shows how the use of semi-trusted hosts and mobile agents can provide for a trade-off between communication overhead and trust in secure distributed computing. There is no need for one generally trusted site, nor does the program code have to be endorsed by all participants. The trusted execution sites are generic and can be used for a wide variety of of applications. The communication overhead of secure distributed computing protocols is no longer prohibitive for their use since the execution sites are located closely to each other.

References

1. M. Abadi and J. Feigenbaum, "Secure circuit evaluation, a protocol based on hiding information from an oracle," Journal of Cryptology, 2(1), p. 1–12, 1990
2. C. Cachin, J. Camenisch, J. Kilian and J. Müller, "One round Secure Computation and Secure Autonomous Mobile Agents", submitted to ICALP 2000.
3. D. Chaum, I. Damgård and J. van de Graaf, "Multiparty computations ensuring privacy of each party's input and correctness of the result," in *Advances in Cryptology—CRYPTO '87 Proceedings* (Lecture Notes in Computer Science, Vol. 293), ed. C. Pomerance, p.87–119 , Springer-Verlag, New York, 1988
4. B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan, "Private information retrieval," Proc. of 36th IEEE Conference on the Foundations of Computer Science (FOCS), p. 41–50, 1995
5. R. Cramer. "An introduction to secure computation", in LNCS 1561, pp 16–62, 1999.
6. S. Even, O. Goldreich, A. Lempel. "A randomized protocol for signing contracts." Communications of the ACM, vol. 28, 1985, pp. 637–647.
7. M. Franklin, "Complexity and security of distributed protocols," Ph. D. thesis, Computer Science Department of Columbia University, New York, 1993
8. M. Franklin and S. Haber, "Joint encryption and message-efficient secure computation," Journal of Cryptology, 9(4), p. 217–232, Autumn 1996
9. O. Goldreich, S. Micali and A. Wigderson, "How to play any mental game," Proc. of 19th ACM Symposium on Theory of Computing (STOC), p. 218–229, 1987
10. O. Goldreich, R. Vainish. "How to solve any protocol problem: an efficiency improvement", Proceedings of Crypto'87, LNCS 293, pp. 73–86, Springer Verlag, 1987
11. S. Loureiro and R. Molva, "Privacy for Mobile Code", Proceedings of the workshop on *Distributed Object Security*, OOPSLA '99, p. 37–42.
12. G. Neven, F. Piessens, B. De Decker, "On the Practical Feasibility of Secure Distributed Computing: a Case Study", to appear in Proceedings of WCC2000.

13. N. Nisan, "Algorithms for selfish agents", *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, Trier, Germany, March 1999, p. 1–15.
14. T. Sander and C. Tschudin, "On software protection via function hiding", *Proceedings of the second workshop on Information Hiding*, Portland, Oregon, USA, April 1998.
15. T. Sander and C. Tschudin, "Towards mobile cryptography", *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, Oakland, California, May 1998.
16. T. Sander, A. Young, M. Yung, "Non-Interactive CryptoComputing for NC^1", preprint.

Nomadic Users' Support in the MAP Agent Platform

Orazio Tomarchio[1], Lorenzo Vita[1], Antonio Puliafito[2]

[1] Dipartimento di Ingegneria Informatica e Telecomunicazioni, Università di Catania
Viale A. Doria 6, 95025 Catania - Italy
E-mail: {tomarchio,lvita}@iit.unict.it

[2] Dipartimento di Matematica, Università di Messina
C.da Papardo - Salita Sperone, 98166 Messina - Italy
E-mail: apulia@ingegneria.unime.it

Abstract. Future communication systems promise to offer a wide variety of highly sophisticated and personalised services over the widest possible coverage area. Users expect to be able to use their subscribed personalised services at any place and any time, transparently and independently of the underlying network technology. Mobile agent technology is seen as a very promising approach to deal with distributed computing and user mobility. In this paper we present an agent-based approach for dealing with users' mobility. The system has been implemented for a mobile agent platform developed at University of Catania, which has been enhanced in order to allow the user to access network services in a mobile environment.

Keywords: Mobile agents, Nomadic computing, Mobile communication.

1 Introduction

The continuous progress in the area of wireless technology and communication networks has enabled the creation of scenarios where users (by means of portable devices) can access several information and services, independently from their location. Besides, the rapid development in personal computing has allowed the design and implementation of more versatile and efficient nomadic devices, thus making them a valid tool for users who frequently need to move from a place to another. The development of distributed applications that can be used in nomadic computing causes considerable issues, due to the restrictions involved in the use of nomadic devices. The mobility of such devices implies that they can connect from different access points throughout the time, and should remain connected even if the user is moving. Their connection to the rest of the network mainly consists of wireless links that are not always reliable, since limited bandwidth and latency characterize them. Such devices are therefore subject to sudden connection failures. For these reasons, the applications currently used

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 233-241, 2000.

by an ordinary user in personal computing cannot be (in general) used in mobile computing. In fact, such applications cannot fully meet the mobile user's requirements, since they have not adequately been developed for taking such limitations into consideration. Specific mechanisms therefore need to be implemented, that can adequately support the user's mobility and assure him/her to access distributed resources independently from his/her location and from the type of (either wireless or fixed) network he/she is connected to.

This aspect is important when considering third-generation mobile systems like UMTS, which implement a single integrated system where the user needs to access the services in a simple and uniform way in any environment. Since UMTS will support wireless communications with higher bandwidth than the present wireless systems, developing and providing advanced services - now limited to fixed networks - will be possible. The availability of such services will be assured by using several access technologies, including fixed, radio, and satellite networks. A key concept for the third-generation networks is the VHE (Virtual Home Environment) [2], which has been introduced into the standardization process for the provision and delivery of personalized services across network and terminal boundaries with the same look and feel. The basic idea is that a user, after subscribing a service, can personalize it and access it while roaming in different networks and (within the capabilities of the terminal) from different terminal-devices. While the VHE has been identified as a powerful and necessary concept for the future integrated service environment, little work has been done to date for its precise definition and validation. Following this research line, we have started to investigate the role that mobile agent technology could have in the telecommunication field.

In this article, a framework for supporting nomadic users, based on the mobile agents technology, will be presented. This framework has been designed and implemented for the MAP [3] system, which is a platform for the development and the management of mobile agents, implemented at University of Catania [13]. The rest of the paper is organized as follows: in Section 2 we will discuss about benefits that mobile agent technology can produce in the scenario that we have described before. Then, in Section 3, we will describe the main characteristics and the implementation strategies used for adding such features to the MAP system. Finally, in Section 4, we present a case study applied to the information retrieval of hotels information. Section 5 presents our conclusions and hints for future research work.

2 Supporting mobile users with mobile agent technology

In this paper, by mobile agent we mean an independent software module acting on behalf of a person or an organization. In order to complete its task, a mobile agent may migrate among several hosts of a network [7, 10]. Even if nowadays there is not a single definition of "mobile agent", the one provided above is

[3] MAP is available at http://sun195.iit.unict.it/MAP

generally accepted, and is useful for making a distinction between our area and the one of intelligent agents, where the attention is focused on the AI mechanisms integrated in such modules [3]. Thanks to its characteristics, the mobile agents technology can be considered a valid enhancement of traditional distributed programming techniques, and an effective tool in mobile computing [4, 1]. In our reference scenario, the user submits a specific request through a mobile agent. The mobile agent, while performing the service requested, may migrate in the network, in order to access the resources it needs. Then it returns to its original node, where it will show the results of its action to the user who triggered it. Thanks to the agents' paradigm, a user can (if he/she desires to) disconnect from the network waiting for the agent to end its task, and reconnect in a second time (not necessarily from the same location) only for obtaining the results from the agent. Besides, the agent may have been instructed for returning the results on different devices (e.g. as an SMS on a mobile phone), or in different formats (e.g. through e-mail messages). The use of mobile agents therefore assures the continuity of a work "session" even after the user who started it has disconnected. In fact, during the disconnection period the session continues in the agent's execution, in a way that does not depend either on the user or on his/her current location. Furthermore, since the agent does not need to continually interact with the mobile device, its execution is in no way affected by sudden connection failures, thus enabling the applications using this technology to benefit from the continuity of the service.

Despite the impact that agents technology might cause on the future systems of mobile communication [6], little work has been made in this area. The support for a user's disconnection is provided in [5], but in that paper no reference is made for the different types of terminal that the user may use. Similar aspects for our approach have been recently provided in [9]; in these cases, the attention is focused on third-generation mobile networks.

3 Mobility support implemented in MAP

In this section we present the support for mobile users that has been designed and implemented for the mobile agent platform MAP. After a short description of the MAP basic architecture and of its characteristics, we will present the infrastructure created for supporting the continuity of service in case the user disconnects from the system, and if different formats are used for adapting the contents to the user's terminal

3.1 The MAP platform

The mobile agent system MAP [13] provides all the basic tools for the creation and the management of agents, and for their migration and communication. In fact, the platform enables to create, run, suspend, wake up, deactivate and reactivate agents, to stop their execution, and to make them communicate and migrate through the network.

MAP is also equipped with a simple graphical interface that facilitates the access to the above mentioned management functions. The MAP platform has been made compliant with the *MASIF* specification [8, 11]; this way, each MAP platform can accept agents coming from different platforms (also complying with MASIF), and can make them run, allowing them to access the methods needed for their management. Besides, the same way, a MAP agent is allowed to migrate to other platforms that can support it, and can run there. According to the MASIF specifications, the different agent servers are grouped in *"regions"*, where an appropriate entity (*Region Registrator*) maintains the information concerning the agents and the agent systems present. Figure 1 shows the logical organization of the different MAP servers.

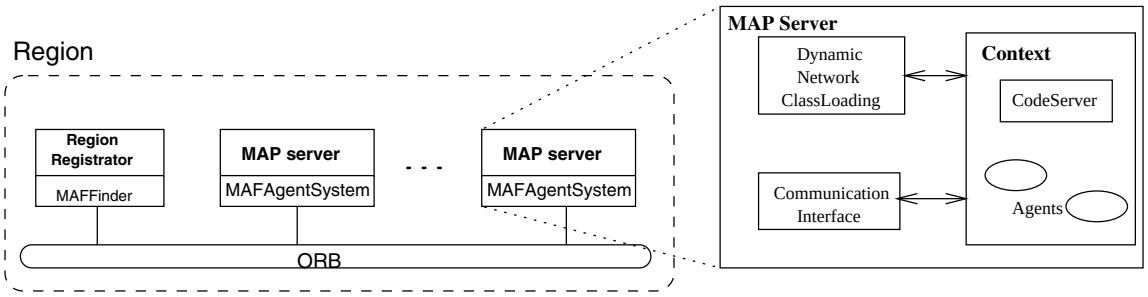


Fig. 1. Architecture of the MAP agent platform

Security is the other very important aspect - present in MAP - for the development of useful agent-based applications. An adequate security model has been implemented in MAP [12], which considers the issues of authentication and authorization, in order to protect - on one hand - the hosts from the agents, and - on the other hand - the agents from the other agents and from the attacks coming from the network. *Security policies* can be defined for specifying the conditions according to which the agents can access host resources. The techniques used are based on public key encryption, in which each user has a pair of public and private keys, with which he/she can prove his/her identity.

3.2 Users' mobility support

The support of users' mobility, which has been specifically developed and integrated in the MAP platform, consists of monitoring the users connected in a *"region"*. This way, when an agent ends its execution, it can be transparently and directly sent to the MAP server which the user is currently connected to (this server might not be the same as the original one). If the user is temporarily disconnected, the mechanism we have developed temporarily deactivates the agent on the MAP server where it is running, waiting for the owner user to reconnect to the system. Once this action is done, the user is notified about all of his/her agents that were *"deactivated"* during his/her disconnection. He/she can therefore decide whether to awake some of them, or not. Once an agent is

reactivated, it automatically migrates to the server which the user is connected to, where it can finally show the user its results.

The solution developed in MAP consists of introducing an appropriate software module in each region. The task of this module, which is called *Lookup Server*, is to keep track of all the users connected to MAP servers belonging to the region, and of all the deactivated agents. When an agent ends its execution and requests to return to its original node for communicating the results of its action to the user who triggered it, the system starts the mechanism that has been appropriately developed for keeping track of the user.

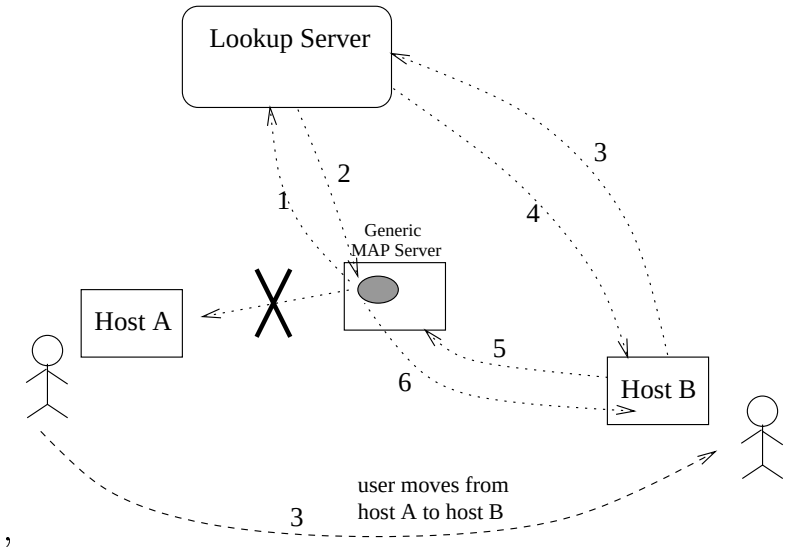


Fig. 2. Supporting mobile users in MAP

The operation of the system, as well as the role played by the Lookup Server, is shown in Figure 2. The steps that are done by the system, in order to satisfy a mobile user's requests, are described below.

When a User Agent has to go back to the user host where it has been started (Host A), (1) the system asks the Lookup Server whether the user is still connected to the original host. Two situations may now occur: either the user is connected a MAP server, or not. In the first case, once the address of the MAP server, which the user is currently connected to, has been determined, the agent is directly sent to that server, so that it can correctly provide the user with the information it has retrieved during its execution. In the second case, (2) the system deactivates the agent on the MAP server where it was executing, and the Lookup Server stores such information. As soon as the user connects to any MAP server (3), the Lookup Server checks whether some agents are associated to that user and are temporarily deactivated in the region and, if there are, (4) notifies the user about their existence, by means of an appropriate graphical interface. The user can select agents from the list, and (5) request to reactivate them. The reactivated agents are (6) automatically sent to the MAP server, which the owner user is connected to, so that they can finally notify him/her

with the retrieved information.

The Lookup Server has been implemented as a CORBA object. It is activated as soon as the Region Registrator is activated, and is present in each region. The following IDL interface defines its service:

```
interface LServer {
    string addUser(in string uid,in Masifname loc);
    void addAgent(in string uid,in Masifname agn,in string cl,in Masifname agl);
    void removeUser(in string uid);
    void removeAgent(in Masifname agname);
    Masifname getUserLoc(in string uid);
    Agelems getUserAgents(in string uid);
}
```

It includes all the methods needed for keeping track of users in a region: the actual implementation of this object might be done in different ways, in order to consider specific application needs such as effectiveness and/or scalability. The choice of designing this component as a CORBA object, simplifies (within the agents platform) the issues linked to the location of this component, which are therefore transferred to the ORB of CORBA.

What we have described before enables a user to submit his/her request (as an agent), to disconnect from the system, to move within the network, and then receive the results at a second time. However, the information and communication domains is gradually converging, in an evolutionary path characterised by Internet-based access networks and high capacity terminals augmented with integrated multimedia service access capabilities, which can take advantage of the increased network capabilities and emerging service features available to the user. Considering this evolution, future communication systems ambitiously promise to provide a wide variety of highly sophisticated and personalised services over the widest possible coverage area. Furthermore, users expect to be able to use their subscribed personalised services at any place, transparently and independently of the underlying network technology. Ideally, a given service should always appear to be the same from the user's perspective, regardless of the access point currently in use and of the physical realization of the service which may differ from one provider to another. This concept is addressed by the Virtual Home Environment (VHE) notion, whose main feature is that the customized environment will be following the user while he/she is roaming within different networks and registering at different terminals.

Following this research line, we have started to develop some features that enable the interaction between the agents' platform and the user with other ways of communication: e-mail and mobile telephony. Essentially, we wanted to equip the MAP platform with some asynchronous (as well as the synchronous modes we have just described) interaction modes with the user, so to provide a complete and versatile support to mobility. We have developed some modules that enable each agent to communicate both through ordinary e-mail messages and through SMS (Short Message Service), by exploiting the present potentialities of the GSM system. When an agent has to deliver the results of its action, it can be

automatically sent to the correct "mobile position" (if necessary, after a period of deactivation), but it is able to communicate its results also if the user has not the immediate possibility to connect to a MAP server.

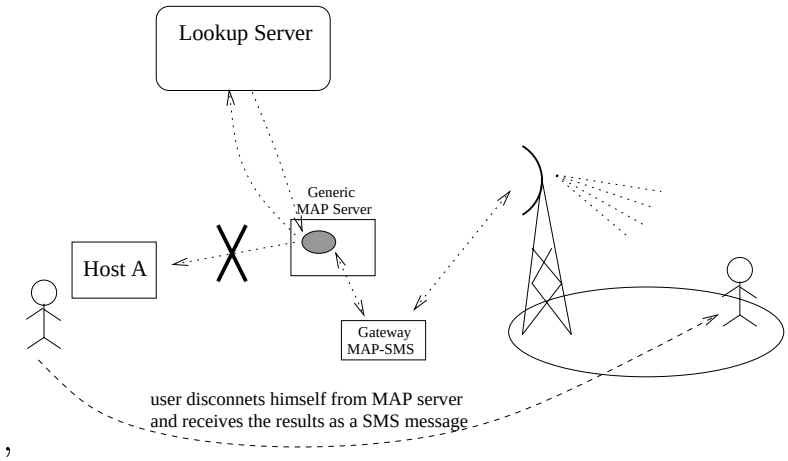


Fig. 3. User receiving results as a SMS message

Figure 3 represents the characteristic according to which a user, after submitting a request, disconnects from the MAP system and receives the results of his/her query directly on his/her mobile phone, as an SMS. Such features provide a first attempt to service adaptability, namely to the automatic adaptation of data to the specific features of the user terminal.

4 Case study: retrieving hotel informations

A simple application has been developed to show and exploit the support to users' mobility. The application is a typical case of information retrieval, where the user might not want to remain connected to the system during the whole search, or wishes to obtain the information at a second time. This application allows searching for information concerning the hotels of a specific town, and provides the details concerning the estimated cost of a room, and the phone numbers needed for contacting the hotel. The user interacts with the system through an appropriate graphical interface (shown in Figure 4), through which he/she inserts the details concerning the information to be searched.

Then such information is sent to an appropriate *SearchAgent*, whose task is that of searching for the requested information, by migrating on appropriate *Information Servers*. When the agent has concluded its search, it can return to the departure site, where it will show the results to the user. This is the *"normal"* behaviour of agents, which happens if the user (after doing the request) remains in the same host until he/she receives the results. However, the system implemented in MAP enables the user to receive the results in any machine he/she connects from, after sending the request. The mechanisms implemented in the

Fig. 4. Graphical interface of the hotel application

MAP can be accessed from by the graphical interface through some checkboxes: *PC, Email, Cellular* (see Figure 4).

By selecting the *PC* checkbox, the platform is enabled for sending the results to any MAP server where the user has connected from, even after a period of inactivity. By selecting the *Email* checkbox (and of course by entering a valid e-mail address) the platform is enabled to return the results to the specified e-mail address. Thus the user, after sending the request, can disconnect from the system, because he/she will receive the results in his/her mailbox. Finally, the *Cellular* checkbox shows the system's capability of interacting with other devices than traditional computers. In fact, through this checkbox, the user enables the system to send an SMS to the GSM phone number indicated in the interface. This message will contain a summary of the information found by the agent, and consisting of the essential information concerning the hotel: name, phone number, and cost. The agent adapts the content of the results to the characteristics of the device through which the user benefits from the results.

5 Conclusions

In this paper we have presented an approach based on mobile agents, for supporting mobile users' needs at the level of the system. We have focused our attention on the main advantages arising from the adoption of a mobile agents paradigm as a middleware technology, and on the transparency of such mechanisms with regard to the user's mobility. We can either refer to a user who moves while the computing resources remain in a wired network, or to a user

who, after doing a task, roams from a communication environment to another one (e.g. from a wired to a wireless network). The flexibility and the naturalness with which agents systems can face the issues related to users' mobility, make them (in our opinion) a good option in this area.

The future work in this area (that will be carried on within the VESPER project) could be the use of agents technology in third-generation mobile networks for the implementation of Virtual Home Environment services.

Acknowledgements: This work has been partially supported by the European Community through the VESPER project (#IST-1999-10825) under the IST programme.

References

1. D.M. Chess, C.G. Harrison, and A. Kershenbaum. Mobile Agents: Are they a Good Idea? Technical Report RC19887, IBM T.J. Watson Research Center, March 1995.
2. ETSI. TS 22.70 Universal Mobile Telecommunication System (UMTS), Service Aspects: Virtual Home Environment (VHE). *UMTS Draft Version 3.0.0*, 1998.
3. S. Franklin and A. Graesser. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agent. *Intelligent Agents III, Springer-Verlag*, pages 21–35, 1997.
4. A. Fuggetta, G.P. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transaction on Software Engineering*, 24(5), May 1998.
5. R. Gray, D. Kotz, S. Nog, D. Rus, and G. Cybenko. Mobile agents: The next generation in distributed computing. In *Proceedings of the Second Aizu International Symposium on Parallel Algorithms/Architectures Synthesis (pAs '97)*, pages 8–24, Fukushima, Japan, March 1997. IEEE Computer Society Press.
6. L. Hagen, M. Breugst, and T. Magedanz. Impact of Mobile Agent Technology on Mobile Communication System Evolution. *IEEE Personal Communications*, 5(4):56–69, August 1998.
7. K. Rothermel and R.Popescu-Zeletin Eds.,. Mobile Agents. *Lecture Notes in Comp. Science*, LNCS1219, 1997.
8. D. Milojicic, M. Breugst, S. Covaci, and al. MASIF: the OMG Mobile Agent System Interoperability Facility. In *Second International Workshop on Mobile Agents, (MA'98)*, Stuttgart (Germany), September 1998.
9. F. Bell P. Farjami, C. Goerg. Advanced service provisioning based on mobile agents. Ottawa, Canada, October 1999.
10. V. A. Phan and A. Karmouch. Mobile Software Agents: An Overview. *IEEE Communication Magazine*, 31(7):26–37, July 1998.
11. E. Di Pietro, A. La Corte, A. Puliafito, and O. Tomarchio. Extending the MASIF Location Service in the MAP agent system. In *IEEE Symposium on Computers and Communications (ISCC2000)*, Antibes (France), July 2000.
12. A. Puliafito and O. Tomarchio. Security mechanisms for the MAP agent system. In *8th Euromicro Workshop on Parallel and Distributed Processing (PDP2000)*, Rhodos (Greece), January 2000.
13. A. Puliafito, O. Tomarchio, and L. Vita. MAP: Design and Implementation of a Mobile Agent Platform. *Journal of System Architecture*, 46(2):145–162, 2000.

Keyphrase-Based Information Sharing in the ACORN Multi-agent Architecture

Hui Yu[1], Ali A. Ghorbani[1], Virendra C. Bhavsar[1], and Stephen Marsh[2]

[1] Faculty of Computer Science
University of New Brunswick
Fredericton, NB, E3B 5A3, Canada
{ghorbani, bhavsar}@unb.ca
[2] Institute for Information Technology
National Research Council
Ottawa, ON, K1A 0R6, Canada
steve.marsh@nrc.ca

Abstract. The ACORN provides an agent-based architecture for information retrieval and provision across networks. The main objective of this paper is to present the design and implementation of information sharing in a community of mobile agents in ACORN, based on keyphrase sharing and comparison among agents. In keyphrase-based information sharing, keyphrases and their weights are used to represent user interests and document contents. Agents use these keyphrase-weight pairs to find other relevant agents. Two similarity measures, the substring indexing method and the Cosine measure method, are compared. The ACORN with Cosine measure is found to be more efficient in terms of meaningful information exchange and execution time.

1 Introduction

Information is playing an increasingly important role. Great changes are taking place in the area of information supply and demand due to the widespread application of computers and the exponential increase of computer networks such as the Internet. The main problems we are facing right now are how to extract relevant, useful, and interesting information from many diverse sources and how to distribute our information to relevant people. Currently, two different technologies are commonly used to address the information demand problem, but fewer systems exist for distributing information. The existing solutions for the information demand problem can be either information retrieval, such as is used in the current crop of Web search engines, or information filtering, for example in systems such as SDI (Selective Dissemination of Information). One embodiment of the information filtering technique is the software agent. Software agents exhibit a degree of autonomous behaviour, and attempt to act intelligently on behalf of the user for whom they are working. Agents maintain user interest profiles by updating them based on user feedback. Examples of such systems are

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 243-256, 2000.

the Kasbah framework for e-commerce [5], and matchmaking systems such as Yenta [3] and ACORN [6].

The ACORN architecture provides an agent-based architecture using community-based approaches for information retrieval and provision across networks. It is based on the assumption that a mixture of consumer pull and producer push, coupled with a tight control of information spread, will allow people to keep up-to-date with topics, and will allow the producers of information to get their information in a timely fashion to those who will find it relevant. The agents in the system are autonomous. They make their own decisions about what to do based on information they receive from their creators and from the data they get from other agents of community. One of the ACORN's goals is to facilitate an architecture whereby social networks of computer (ACORN) users could share information based on 'people chains.' ACORN's previous incarnations, however, used a very rudimentary approach to keyphrase matching among agents, in that we had a simple boolean, exact matching algorithm: if two keywords or phrases matched exactly, the agents containing those keyphrases would be deemed similar. This situation is clearly unsatisfactory for all but a research setting. In this paper we extend ACORN in a direction which allows it to consider two agents in a more meaningful fashion, using agent keyphrases.

The work reported in this paper proceeded in two stages. The first was to produce a prototype architecture to test certain methods of information sharing and agent grouping. The second was to incorporate into ACORN the salient points of this prototype. In both cases, extensive testing was performed using both synthetic (machine generated) and 'real world' (obtained from journal articles) data. In order to carry out testing, we have introduced earlier a novel concept of multiple autonomous virtual users [1]. The main objective of this paper is to present the design and implementation of information sharing in a community of mobile agents in ACORN based on keyphrase sharing and comparison among agents. This paper is organized as follows. An overview of ACORN is given in the following section. The basic concepts of keyphrase-based information sharing are introduced in Section 3. In Section 4, two keyphrase-based similarity measures are introduced and their performance is compared. Finally, conclusions are given in the last section.

2 ACORN

ACORN is a multi-agent based system which uses the concept of 'information as agent' together with an application of Stanley Milgram's Small World Problem [7] to route information around networks (or communities) of people. Information in ACORN's context is anything that can be transported electronically, such as documents, in whole or in part, queries, images, sounds, and so on. In ACORN, any piece of information is represented by a mobile agent. The implementation of ACORN referred to in this paper uses two kinds of mobile agents: *InfoAgent* and *SearchAgent*, and we will discuss these further below.

In order to provide the infrastructure for the mobile agents to perform their tasks, several static agents also exist. These are the *Server*, the *Client*, and the *Café*.

The premise of ACORN is that information can find its own way to relevant people. By 'relevant,' we mean that the person is interested in the information, or is able to help with the query presented to them. In ACORN, as in the Small World Problem, the concept of Degrees of Separation is used. Thus, the mobile agents follow chains of people- (and/or agent-) based recommendations in order to find those which are 'relevant.' We are using an extended form of 'who knows who knows what,' in other words. ACORN at present uses keyphrases to ascertain relevance. Thus, if an agent's keyphrases match a user's, we can say that for that keyphrase, the information the agent is carrying is relevant to the user.

The remainder of this section details the ACORN infrastructure, the agents that make it up, and finally discusses agent-based information sharing in ACORN, which is performed exclusively in Cafés.

2.1 Static ACORN Agents

The static agents in ACORN architecture are used to control mobile agents migration, communication, information sharing, and so forth. A brief description about each static agent in ACORN is given below. In order to facilitate understanding, Figure 1 shows an overview of the architecture at a single site.

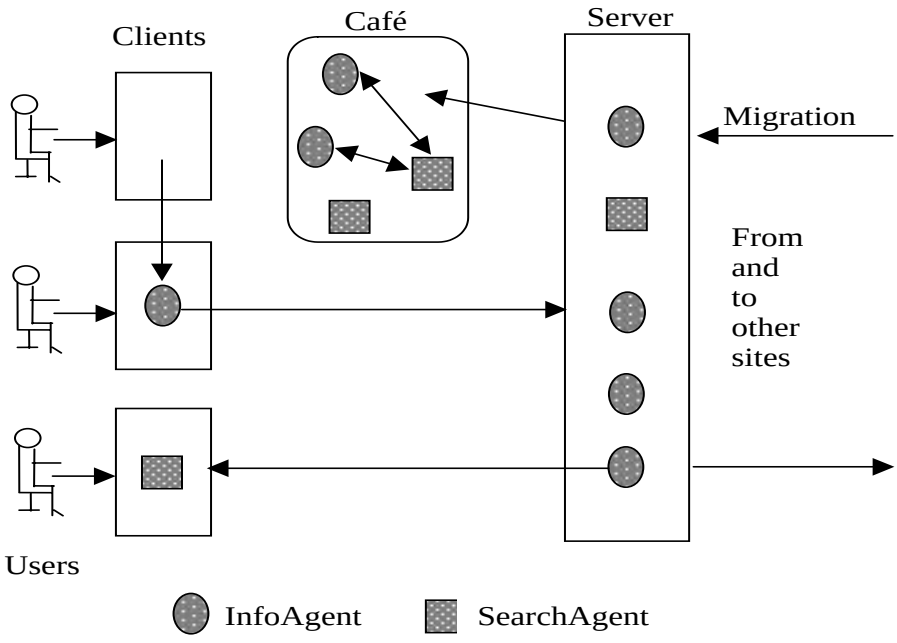


Fig. 1. Overview of a single ACORN Site

Client ACORN's Client serves a dual purpose. Its primary role is to provide a user interface to the ACORN system. It allows the creation, browsing, control, tracking, and deletion of incoming and outgoing ACORN agents, and in this sense is much like any email client.

The second role of the Client is to maintain a user profile and perform information handling tasks. In this role, the Client acts as both a filter to incoming information (agents) and a form of 'community browser.' In its filtering tasks, the Client can scan, prioritise and route incoming agents according to the preferences based in the user profile (which is controlled by the user and includes priority information, user interests, contact information, and so forth). For more information on the Client and user profile, see [6].

As a community browser, the Client tracks incoming agents and files information about these agent's owners (that which is made public by the owner's Client on agent creation - see [6]). In addition, the Client's user can input similar information, which takes the form of 'person X is interested in topics $y, z, \ldots$'. This information can be used at agent creation to suggest possible recipients for an agent. Over time, then, the Client can build up a fairly sophisticated world view of the communities its owner belongs to. Again, this can be used in filtering, prioritising, and recommending tasks.

Server Much like a Web server, the ACORN Server resides at the point of entry from a network to a site. All mobile agents must enter the site through the Server. While the Client acts as a user-controlled information filter, amongst other things, the ACORN Server is a site filter whose primary task is to protect the site and decide what kind of mobile agent is allowed in. It also controls mobile agent migration carried out via server-to-server communication. In addition, it acts as a permanent repository for Client and resident mobile agents at a site, in order to achieve persistence in agents.

When a mobile agent arrives at a site, the Server saves its state. Clients communicate their state to the Server whenever they are started up and at specified intervals while they are running. The Server stores the Client data and can augment it with messages for the user if any are sent. For more information, see [6].

Café The Café is a virtual meeting place for ACORN agents. In the Café, at prescribed times, all agents present give their community and personal data to the Café manager, which compares all data, sharing out any relevant community data to agents who would find it of use, before sending out the agents. The net result is that, on exit, agents may well know more about the community than when they came into the Café.

For example, if two agents enter the Café, one representing a document about programming in Java (keyphrases: OOP, Objects, Java, etc.), the other representing a query about Object Oriented programming (keyphrases: Objects, OOP, Software Engineering, etc.), the Café will match the keyphrases and take the relevant user information from each agent, giving it to the other. The net result is

that the first agent will now visit the second agent's owner to present its information, which may be of relevance, and the second agent will visit the first agent's owner, hoping to learn more. Of course, we envisage that many more than two agents will be present in each Café Information Exchange, with a corresponding increase in the utility of exchanged information. At present, each agent will visit the Café at every site it visits (a site may have from zero to many Cafés, each of which could be dedicated to a specific genre of information, but the decision on which to visit is the agent's), thereby spreading and gaining relevant information. In this manner, the Café automates community based information sharing for agents.

2.2 Mobile ACORN Agents

The ACORN system currently has two types of mobile agents: *InfoAgent* and *SearchAgent*. These agents perform the task of moving information or queries from site to site to relevant readers.

The InfoAgent An InfoAgent is generated when any item of information is created for distribution. On creation, the InfoAgent is given a list of keyphrases for the information (which the Client obtains automatically if possible) and can be given additional metadata. This metadata is stored in a Dublin Core element set (see [10]). The InfoAgent carries this metadata information, and to preserve bandwidth, a reference to the information itself — in the present incarnation of ACORN, the information itself is not carried. This not only preserves bandwidth but also allows a certain degree of additional security control over the information itself. As will be noted below, however, the SearchAgent can carry a complete query.

In addition, the agent is given a list of possible recipients — generated either by the Client from given keyphrases, and/or suggested by the user. Note that this information in fact need not be given — the agent can find its own recipients via the Café mingling process.

Once created, the InfoAgent migrates around the ACORN network delivering its information, and finding (via both Café recommendations and via recommendations from users it delivers itself to, in the manner of email forwarding) other users who would find it of interest. As a technical point, in ACORN the code of the agents is not mobile, only the metadata and associated recipients lists (the AgentCore) is. The migration process involves sending a core to a new Server, where a new agent is instantiated to handle the core. This decision was taken in the interests of security on the Server side, but carries with it ramifications for the agent, which will need to be addressed.

The SearchAgent The SearchAgent is a specialised form of InfoAgent which carries a query. At present, the SearchAgent behaves much as the InfoAgent does — both in terms of creation, recommended recipients (those who can help with this query) and migration. However, we are investigating opportunities to

allow the SearchAgent to become more general purpose, enabling migration to and searches of web sites or web indexes.

In the SearchAgent, the query can be expressed in a very powerful form, as the whole of the Dublin Core element set is available to specify query requirements. However, since the Dublin Core elements are all optional, the query can also be expressed simply as a set of keyphrases or a natural language sentence (in the latter case, however, the Café-based sharing will not be available, since keyphrases are used at present in the Café).

2.3 Migration and Interactions

Migration in ACORN is very important to the system. Without migration, one agent cannot reach other agents to interact with them. Hence information cannot be shared. In fact, as was mentioned above, only information or queries carried by agents migrate. The agent's codebase does not migrate.

When a mobile agent wants to migrate to another site, it informs the Server at the present site of this fact. The site's Server first ascertains that the mobile agent is allowed to move from this site. If so, the agent's state is passed to the remote Server via ACORN Server-to-Server protocol. At the other site, the remote Server decides whether or not the migration can take place. If not, it informs the mobile agent, and the agent reorganizes its migration strategy (usually, going to the next person on its list at another site).

When an agent arrives at an ACORN site, two types of interactions between agents take place. The first interaction is that the mobile may have a specific person (his/her Client) to contact. In this way, information can be shared between the person and the mobile agent. The second type of interaction takes place between mobile agents in a Café. Once all the recommended users on a site have been visited, the agent should have an enlarged list of users to visit, and possibly a set of query responses (which it can carry with it or send back to its own user). Either way, it can be seen that the agent obtains more community information as it migrates, amongst other things, and that this information can be made use of at the end of the agent's life (when it returns home to its user's Client to upload the community information it has gathered).

3 Keyphrase-based Information Sharing

People create agents to work for them, but their purposes may not be the same. For example, some people may create agents for searching for information, while others may create agents for distributing information to relevant people. One of the important aspects of the information retrieval and provision is to share information among agents. In information sharing, an agent can receive information from and provide information to other relevant agents. In the keyphrase-based information sharing, keyphrases are used to represent user interests as well as contents of the document carried by the agent. Agents use these keyphrases to find other relevant agents to exchange information.

In ACORN, the information sharing takes place in the Café. The job of a Café is to facilitate information exchange and interaction among relevant agents. There is a Café manager that performs many functions. The Café manager ensures that the agents have specified format (see Section 4 for details). It can control the maximum number of agents allowed at a given time in the Café. When an agent arrives, the Café manager checks if the Café has room for the newly-arrived agent. The agent has to wait if the Café is full. Agent leaves the Café after it has stayed for a specified period of time.

A blackboard is used to keep the information of a certain number of agents after they have left the Café. The blackboard allows other agents to get the information of an agent, even though they could not meet the agent in the Café. The information of an agent is kept on the blackboard for a specified amount of time.

As soon as an agent enters a Café, it immediately starts interacting with other agents in the Café. The aim of interacting with other agents is to find agents with similar interests and exchange information with them. The agent may also receive relevant agent information by inspecting the blackboard in the Café or through other agents recommendations. Since agents have their own specific tasks, they each behave differently while they are looking to find other relevant agents to share information with.

In ACORN, agents share information whenever there is at least one keyphrase match [6]. It is obvious that using such an exact keyphrase matching, an agent collects large amount of information from other agents and finally delivers this information to its owner. Therefore, this is not an *efficient* way of searching and distributing information. This matching method makes it almost impossible for people to extract *relevant*, useful, and interesting information from information sources. In the same vein, it is also impossible for people to distribute their information to other *relevant* people. In order to alleviate these problems, we consider two similarity measures for keyphrases in the following section and carry out their evaluations.

4 Keyphrase-based Similarity Measures

Similarity measures are widely used in the information retrieval (IR) community and are sometimes referred to as the matching functions, the correlation coefficients, or the selection algorithms. In IR, similarity measures are the mechanisms through which the retrieval software makes a comparison between document and query representations to effect retrieval [4]. A similarity measure is any function that assigns a value of matching coefficient to a pair of vectors. Each vector includes a set of attributes that characterizes an entity. Similarity measures have been used to cluster documents and determine similarity between a query and a document. When the similarity measures are used to cluster documents, they are designed to quantify the likeness between documents. If one assumes that it is possible to group documents in such a way that a document in one group is more like the other members of that group than it is like any object outside that

group, then a cluster method enables such a group structure to be discovered. When these measures are applied to determine the similarity between a query and a

document, they serve in matching or ranking. In this section we adapt two similarity measures used in IR to share information amongagents in the Café.

4.1 Similarity Measures

There are number of matching coefficients techniques that can be used to measure the similarity between keyphrases of two agents. We consider two of them in the following, namely the substring indexing method [6] and the Cosine measure method [9].

Let a_i represent i-th agent and $K^{(a_i)}$ represent the set of keyphrases associated with agent a_i. Further, $K^{(a_i)} = \{k_1^{a_i}, k_2^{a_i}, \ldots, k_m^{a_i}\}$ where $k_j^{a_i}$ represents the j-th keyphrase of agent a_i. Adapting the substring indexing method, the similarity of two agents, S_{a_1,a_2}, is the proportion of common keyphrases carried by the two agents and is given as follows:

$$\mathbf{S}_{a_1,a_2} = \frac{2.|K^{(a_1)} \cap K^{(a_2)}|}{|K^{(a_1)}| + |K^{(a_2)}|} \tag{1}$$

where $|.|$ represents the cardinality of a set. $|K^{(a_1)} \cap K^{(a_2)}|$ is the number of keywords shared by both agents. S_{a_1,a_2} is equal to 1.0 for agents with identical sets of keyphrases and 0.0 for agents with no common keyphrases.

In the above similarity measure, all keyphrases are considered to be of the same importance. However, in reality in most of the cases, the keyphrases would have different weights representing somehow their importance. Let $w_j^{(a_i)}$ represent the weight of the keyphrase $k_j^{(a_i)}$ and $\mathbf{K}^{(a_i)}$ represent the keyphrase vector defined as follows:

$$\mathbf{K}^{(a_i)} = \left\{ \left(k_1^{(a_i)}, w_1^{(a_i)}\right), \left(k_2^{(a_i)}, w_2^{(a_i)}\right), \ldots, \left(k_m^{(a_i)}, w_m^{(a_i)}\right) \right\} \tag{2}$$

With the Cosine measure method we define the similarity of two agents as

$$\mathbf{S}_{a_1,a_2} = \frac{\sum w_i^{(a_1)} w_i^{(a_2)}}{\sqrt{\sum \left(w_i^{(a_1)}\right)^2} \sqrt{\sum \left(w_j^{(a_2)}\right)^2}} \tag{3}$$

The Cosine measure method finds the cosine of the angle between two vectors defined in the space of keyphrases; note that each keyphrase represents a unique dimension in this space. S_{a_1,a_2} is equal to 1.0 for agents with identical keyphrase-weight pairs and 0.0 for agents with no common keyphrases.

4.2 Comparison of Similarity Measures

This section describes the evaluation of the two similarity measures explained in the previous subsection, the sub-string indexing method and the Cosine measure

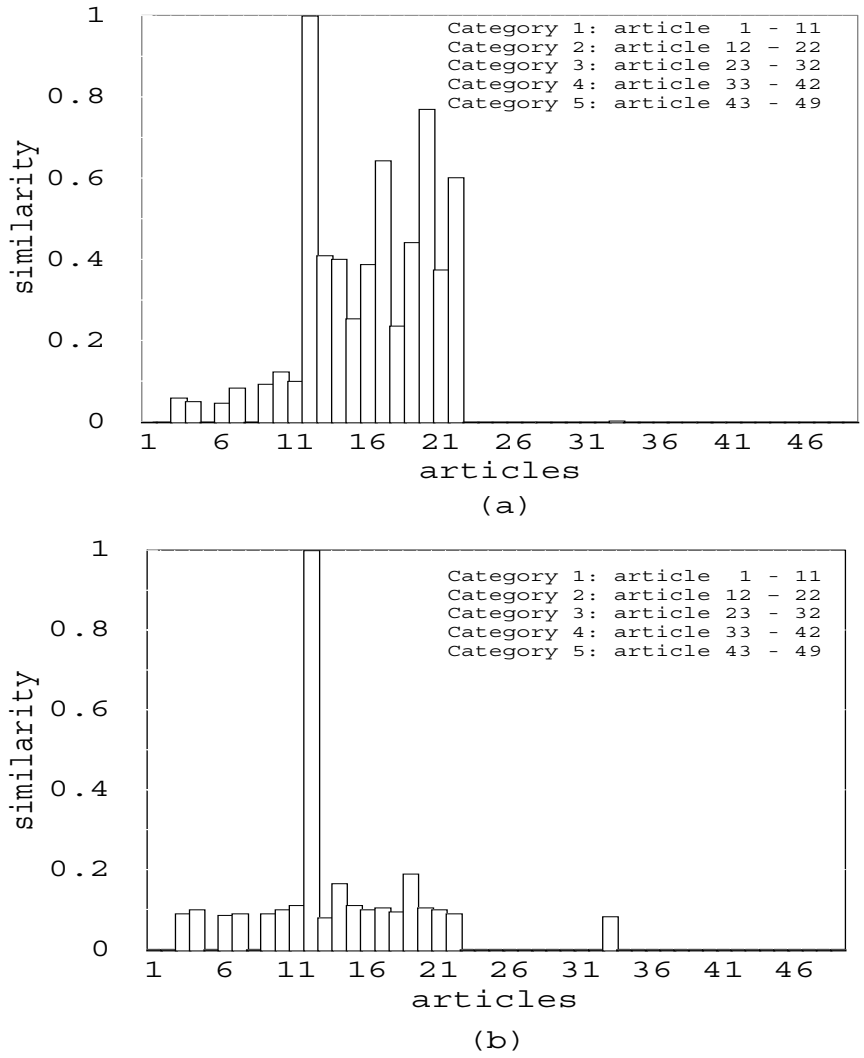


Fig. 2. Similarity of article 12 (from category 2) with all articles: (a) the Cosine measure method, (b) the substring indexing method

method. For comparing the methods we use a sample of 49 article abstracts from technical magazines and web pages. We consider 49 ACORN agents each carrying keyphrase-weight pairs of only one of these articles.

The 49 article abstracts are classified into five categories based on their content: Category 1 (articles 1-11), Category 2 (articles 12-22), Category 3 (articles 23-32), Category 4 (articles 33-42), and Category 5 (articles 43-49). Each category represents a specific topic. The text document indexing software Extractor, developed by the Interactive Information Group of the National Research Council of Canada (Turney, 1999), is used to obtain keyphrases and their weights. The details of these articles and the output obtained using the Extractor are given in [11]. One article from each category was selected and the similarities between this article and the entire set of articles from all categories were calculated using equations (1) and (3). Note that the similarity between two articles is commutative.

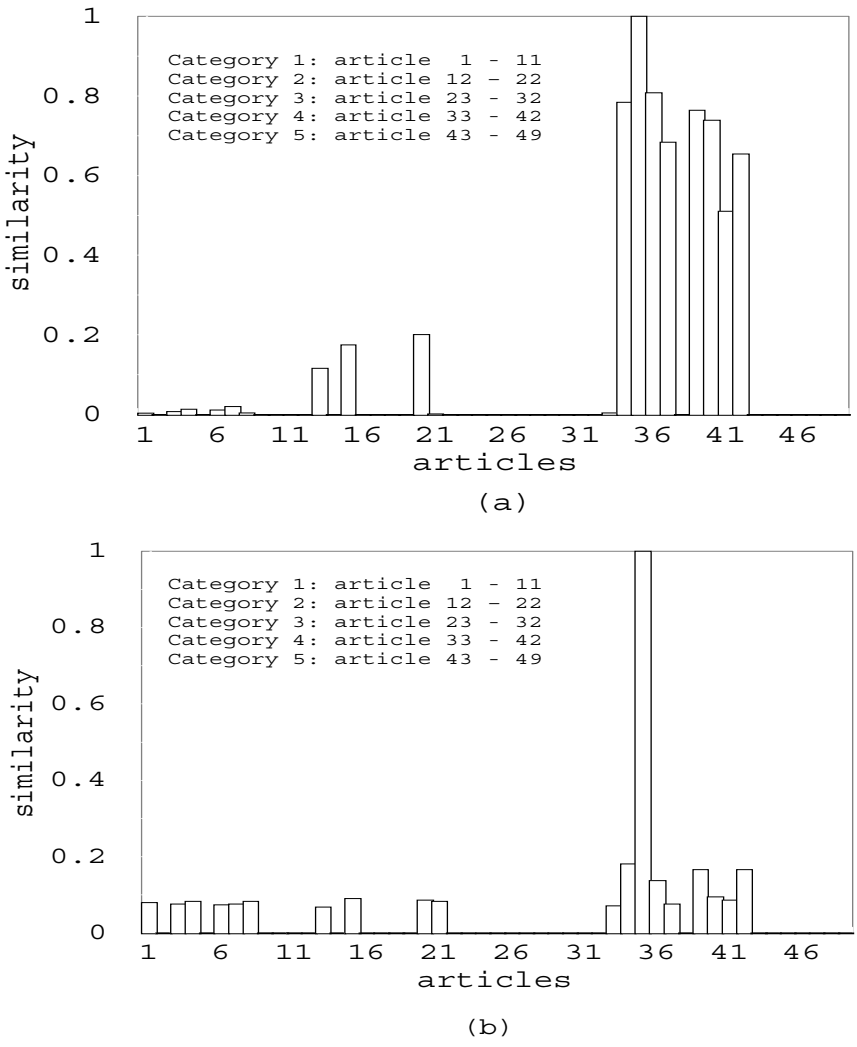


Fig. 3. Similarity of articles 35 (from category 4) with all articles: (a) the Cosine measure method, (b) the substring indexing method

Figures 2 and 3 show the similarity measure results using the Cosine measure method and the substring indexing method. It is seen from Figure 2(a) that for the article 12 of Category 2, the $S_{a12,a12} = 1.0$. Further, the degree of similarity of article 12 with the rest of the articles of its category is much higher than the similarity with articles from other categories. For the substring indexing method, as seen in Figure 2(b), the $S_{a12,a12} = 1.0$. However, there is not much difference seen in the degree of similarity with articles from its own Category 2 as well as Category 1. Thus, the Cosine measure method is found to be superior than the substring indexing method. Similar observations can be made from Figure 3 and additional results given in [11].

4.3 Recall-Precision with Cosine Measure

In information retrieval the precision and recall are commonly used to measure the effectiveness of the retrieval methods. The recall measures the ability of a

method to retrieve useful documents. It is defined as the proportion of relevant material retrieved. The precision, conversely measures the ability of a method to reject useless materials, is defined as the proportion of retrieved material that is relevant to a query. In the following discussion, we adapt the definitions of precision and recall used in the information retrieval to define the quality of information exchange among agents.

The agents in a Café are divided into two sets, the set of the agents, A, that carry relevant information and those that do not, $\bar{A}$. Let a represent the number of agents from A that are selected by the similarity measure method for information exchange. Further, $b = |A| - a$. Let c denote the number of agents also selected from $\bar{A}$ for information exchange and finally $d = |\bar{A}| - c$. Based on these notations, the precision (p) and recall (r) measures are defined as follows:

$$p = \frac{a}{a+c} \tag{4}$$

$$r = \frac{a}{a+b} \tag{5}$$

The above precision and recall definitions were sufficient and appropriate for the early IR systems,

which were merely capable of boolean searching. In the early IR systems, a user's query was expressed as

a boolean combination of keywords, and the systems retrieved the documents matching the constraints

represented by the query [2]. Unlike the early IR systems, our information

sharing system uses vector-space model based on keyphrase-weight pairs. It is used to calculate the degree of similarity between two agents. The owner of an agent can assign a similarity threshold, ST, to his/her agent. An agent a_1 will share information with another agent a_2, if $S_{a_1,a_2} > ST_{a_1}$.

The choice of the threshold will greatly influence the precision/recall of the multi-agent system. For example, when an agent communicates with other agents, if it shares information with a greater number of relevant agents, the system's recall value increases. However, when it shares information with larger number of irrelevant agents, the precision decreases. In order to take this trade-off into consideration,

we evaluate the keyphrase-based information sharing system using average of precision and recall values over the set of 49 articles for different thresholds.

Table 1 lists the average precision and recall values for a test data set (see [11] for details). Figure 4 depicts the average precision versus average recall for various similarity thresholds for the test data set given in Table 1. It is seen that as the similarity threshold is increased, the precision increases whereas the recall value decreases.

4.4 Timing Results

Experiments were carried out to study how the Cosine measure method affects the execution time of ACORN. For each agent a set of keyphrases with ran-

Table 1. Average precision and recall for various similarity thresholds.

	Similarity Threshold									
	0.05	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Precision	0.82	0.9	0.98	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Recall	0.95	0.95	0.93	0.85	0.75	0.64	0.41	0.38	0.21	0.1

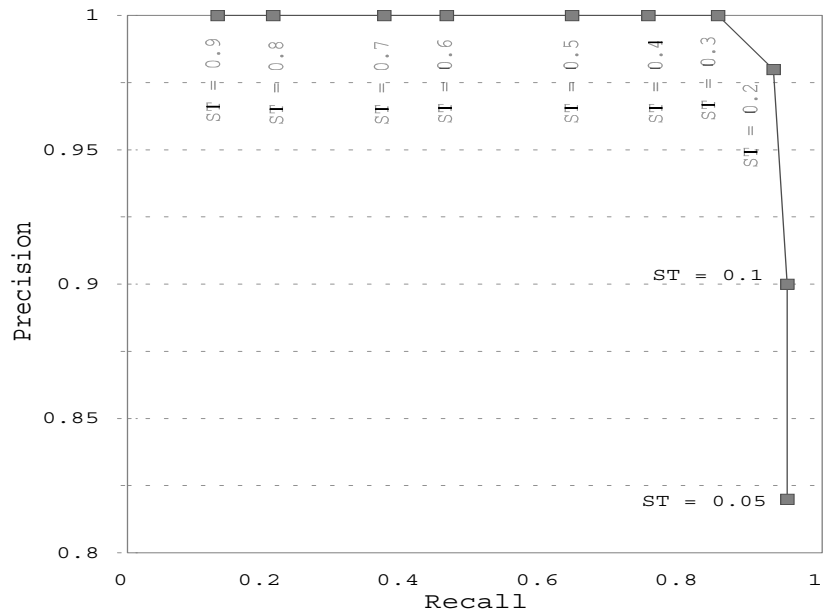


Fig. 4. Average precision versus average recall for various similarity thresholds (ST).

dom weights were generated. We define the *mingle-time* as the time needed for information sharing in the Café. Ten experiments with number of agents ranging from 10 to 100 were conducted for ACORN, with and without the Cosine measure method. Figure 5 shows the experiment results. The execution time is the average of three runs for each experiment. It is seen that the mingle time of ACORN with Cosine measure is smaller than that of ACORN not using the Cosine measure. This is due to selective information sharing done based on the Cosine measure method. The figure also shows that the total execution is reduced when the Cosine measure method is used.

5 Conclusion

In this paper we have presented the design and implementation of keyphrase-based information sharing in a community of mobile agents in ACORN. In keyphrase-based information sharing keyphrases and their weights are used to represent user interests and document contents. The keyphrase-weight pairs are used by agents to find relevant agents. Simulation studies were carried out to compare two similarity measurement methods, the substring indexing method and the Cosine measure method. The Cosine measure method is found

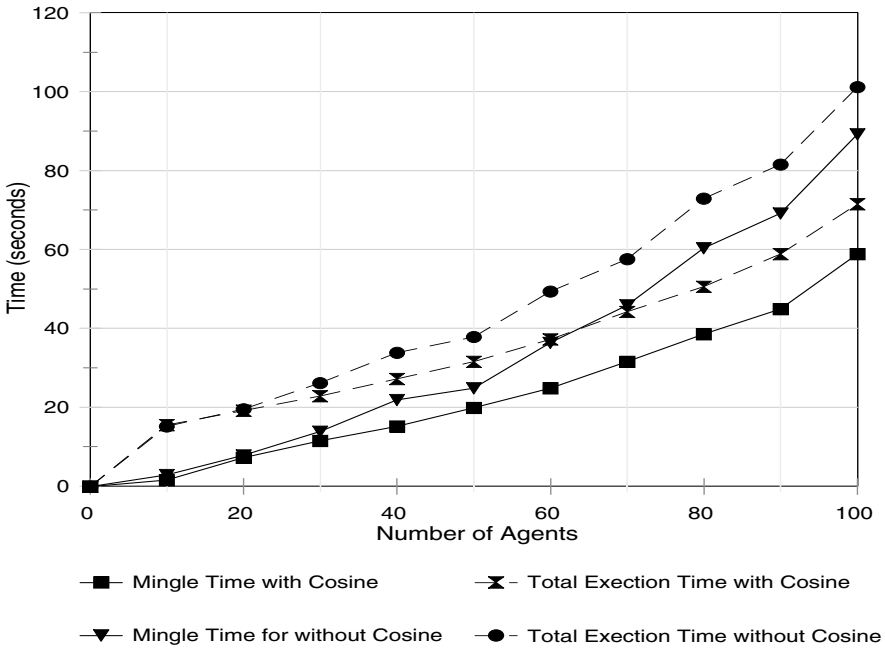


Fig. 5. The execution times for ACORN with and without the Cosine measure method.

to be superior. The Cosine measure method was subsequently incorporated into ACORN. The results obtained shows that ACORN with Cosine measure is faster: this is mainly due to the smaller number of agent interactions. The number of agent interactions in ACORN with even only one an exact match is $\Theta(n^2)$, where n is the number of agents in the Café. In contrast, in the keyphrase-based matching using the Cosine measure method, the number of agent interactions, in the bast case, is $\Omega(n)$ and $O(n^2)$ in the worst case. The results also reveal that more meaningful information exchange takes place with the Cosine measure method.

References

1. Bhavsar, V.C., Ghorbani, A.A. and Marsh, S., "A Performance Evaluation of the ACORN Architecture" , to appear in the Proc. of the 14th Annual International Symposium on High Performance Computing Systems and Applications (HPCS'2000), Victoria, British Columbia, Canada, June 14-16, 2000, http://www.cs.unb.ca/profs/ghorbani/ali/research3.htm/
2. Brüninghaus, S., and Ashely, K. D., "Evaluation of Textual CBR approaches", in Proceedings of the AAAI-98 Workshop on Textual Case-Based Reasoning (AAA-Technical Report WS-98-12), pp. 30-34, Madison, WI, December 1998.
3. Foner, L. N. Yenta: A Multi-Agent, Referral Based Matchmaking System, the First Conference on Autonomous Agents (Agents '97), Marina del Rey, California, February 1997; http://foner.www.media.mit.edu/people/foner/yenta-brief.html/
4. Gerrie, D., *Online Information Systems*, Information Resources Press, Arlington, VA, 1983.
5. Maes, P., Guttman, R.H., and Moukas, A.G. Agents that Buy and Sell, Communication of the ACM, Vol.42, No.3, pp. 81-91, March 1999.

6. Marsh, S. and Masrour, Y. Agent Augmented Community Information - The ACORN Architecture, Proceedings of CASCON'97, Meeting of Minds. November 1997. Available via http://www.iit.nrc.ca/II_public/acorn.html
7. Milgram, Stanley, "The Small World Problem", in Milgram, S. (Sabini, J. and Silver, M. (eds)), "The Individual in a Social World: Essays and Experiments", 2nd Edition. NY: McGraw-Hill, 1992.
8. Morita, M. and Shinoda, Y. Information Filtering Based on User Behavior Analysis and Best Match Text Retrieval, In Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, pp.272-281, Springer-Verlag, 1994.
9. Salton, G. and McGill, M. Introduction to Modern Information Retrieval, McGraw-Hill, New York, 1983.
10. Wibel, S., Godby, J., Miller, E., Daniel, R. "OCLC/NCSA Metadata Workshop Report", 1995. http://purl.oclc.org/dc/
11. Yu, H. "Keyphrase-Based Information Sharing in Multi-Agent Systems", MCS Thesis, Faculty of Computer Science, University of New Brunswick, Fredericton, March 2000.

Agents Based Implementation of Personalised News Delivery Service

Henryka Jormakka[1], Toni Jussila[1], Kirsi Valtari[2],

[1]VTT Information Technology, Otakaari 7 B, Espoo, P.O.Box 1202 FIN-02044 VTT, Finland
{henryka.jormakka, toni.jussila}@vtt.fi
[2]Sonera SmartTrust LTD, Elimäenkatu 17, Helsinki, P.O.Box 425, FIN-00051 Sonera, Finland
kirsi.valtari@sonera.com

Abstract. This paper reports results of development and piloting of an agent based news delivery service. A key interest in the work has been the applicability of mobile and intelligent software agents in telecommunications applications. In order to gain insight and practical experience, the agents were applied to implement personal mobility and service selection functions. Mobile agents were used to provide a roaming user with services subscribed from his local service provider domain. Dynamic service negotiation was implemented by exploiting both mobility and agent intelligence. The news delivery service was integrated, tested and installed in a pilot network, where it was promoted for test usage. This paper presents and discusses the feedback and results received from software developers, end users and the performance measurements made.

1 Introduction

The telecommunications environment is changing towards an open market of services that can be obtained at any time and at any place. In this environment the success of service providers (SPs) will depend on the efficiency of the development of the services, on their accessibility, performance and ability for customisation. In this context software agents offer a new programming paradigm, which raises a lot of expectations to tackle of both information technology and telecommunications, such as mobility, service personalization, or interoperability.

An agent is a software program, which acts autonomously on behalf of a user (which can be a person or a computer system) or acts to achieve a given goal [5]. The roots of the agent paradigm are in the fields of artificial intelligence and human computer interaction as well as object-oriented programming [6]. In the field of distributed artificial intelligence, the research on agent collaboration and interaction lead to multi-agent systems and also to the agent communication languages. In the human computer interaction field, the agents were seen as personal assistants of the user, who would learn the preferences and habits of their owner.

The concept of a mobile agent combines the characteristics of an agent and code mobility. A mobile agent is an agent that can move in the network performing various tasks [2]. The mobile agents may reduce network load and provide asynchronous mode of communication [4]. In addition to this, they might turn out to serve as a more intuitive

E. Horlait (Ed.): MATA 2000, LNCS 1931, pp. 257-269, 2000.

approach for software engineers currently working with remote procedure call paradigm. The agent paradigm has been proposed as a successor of client-server in designing distributed systems. The use of mobile agents requires support from the hosting nodes, which has lead to the development of several agent platforms such as Voyager, Grasshopper, Aglet and Concordia [7], [2]. The agent platforms provide typically agent transfer mechanisms, naming and locating services and control of the agent and they could be seen as middleware, which enhances Java objects into mobile objects.

As the Internet network keeps spreading from offices into homes and mobile phones, it has become apparent that the IP technology is the platform on top of which future network services will build on. As a next step the telecommunications infrastructure may also be built on IP-based technology. The low cost and the endless possibilities of open technology will motivate the telecommunications players to solve the problems of security, charging and best-effort packet transfer. As user mobility has become a major requirement for networks, both fixed and mobile networks should support the end user in accessing his services and data. In a situation where all the services are built on IP-technology, the user should have a service level roaming possibility over all kinds of networks both in office and in public operator administrative domains regardless of the transfer network technology.

In this paper the authors describe a system where the service access in foreign service provider domain was implemented in a similar fashion as in GSM. A concept of visited Service Provider (visited SP), an initial contact point in foreign network environment, was used. Mobile agents were used to support the roaming between SP domains. Additionally agents were exploited in implementing dynamic service selection among competing offers from different service providers. The potential service providers who could be able to provide an instance of optimal service at a given moment are called Candidate Service Providers.

The paper is based on work and results achieved in the context of the EU/ACTS MONTAGE project [1]. The project belongs to the CLIMATE (Cluster for Intelligent Mobile Agents for Telecommunication Environments [3]) initiative within the European ACTS programme.

2 News Delivery Application

For the purpose of the project news delivery service using the agent paradigm was developed. In case of the user terminal the development included the service access and selection logic as well as the interfaces to the personalisation and selection procedures. In the SP domain the service access and provision were implemented with the mobility and dynamic negotiation aspects incorporated already in the software architecture design. The SP functionality included also accounting and charging functions. The content archive node, called content provider node in this text, mapped the architecture to the commercial multimedia server tools and existing news data sources. Each of these domains of the distributed service architecture will be presented in the following.

2.1 Service Access and Selection

Let us consider a scenario where a user has subscribed news delivery service from a local service provider, and wants to use this service while being abroad.

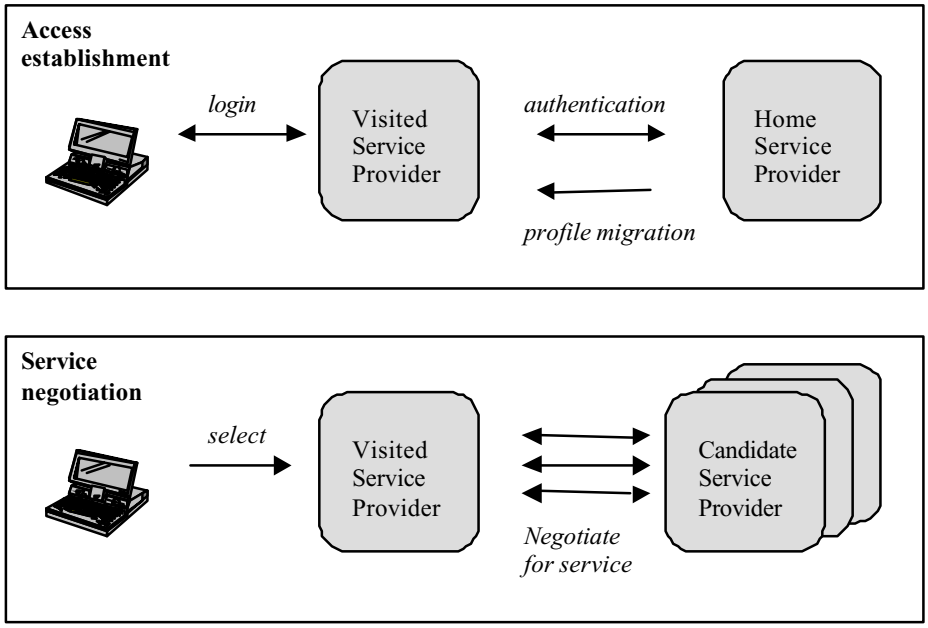


Fig. 1. Communication between various participating service providers within access establishment and service selection.

Fig. 1 points out the interfaces where communication is active during subsequent phases of service access and selection. Here Home Service Provider (Home SP) is a specific SP, to whom a user has a subscription for one or more services. The Home SP has a user specific database called user profile and a contractual position to bill the user for services used by him. It is assumed that there is a single Home SP per service. The user establishes an access session with a local SP, called Visited SP. The user gets authenticated through the Visited SP, which contacts the user's Home SP. In case of positive authentication, a subset of the user's personal profile is copied from home to the visited domain. The subset consists of the list of services that the user may use in the current domain, the user's preferences and the list of service providers that may be considered for the service provision according to federation agreements already established with the home SP. If there are local SP's that have roaming agreements with the user's home, the user may enjoy personalised service provided locally, while at the same time having the possibility to dynamically select a suitable service offer.

Mobile agents were applied in the above scenario to provide a virtual home environment in the visited network and also to implement an efficient negotiation between the SP's to find the most optimal service offer currently available for the user. From this paper's point of view, two mobile agents are of particular importance: User Access Agent (UAA) and Service User Agent (SUA). UAA migrates during the access establishment phase from the user's Home SP to the Visited SP domain to represent there the user during the access session. It is an agent that is responsible for handling the user's profile by first delivering it to the visited domain, and then after the required services are provided to the user, for returning his profile's updates back to the home domain.

Additionally, the UAA creates SUA - a specialised agent that represents the user during the negotiation phase with federated SPs that are listed as candidates for offering the service. In order to perform its task, the SUA created in the visited domain sends its own replicas to all candidate SPs that offer the service requested by the user. The replicas locally negotiate the best offers, using the information obtained from the user profile, some of which may be confidential and should be kept secret to the SPs. The negotiation is done on the basis of service preferences set by the user and market offerings promoted by the SPs. After the negotiations are completed, the SUA at the visited domain collects the results provided by its replicas and makes decision on the best offer from the user's point of view. The decision is passed to the UAA and the SUA terminates. The SP whose service has been chosen will be further called selected service provider. The software components introduced above are described in detail in [10].

2.2 User Interface

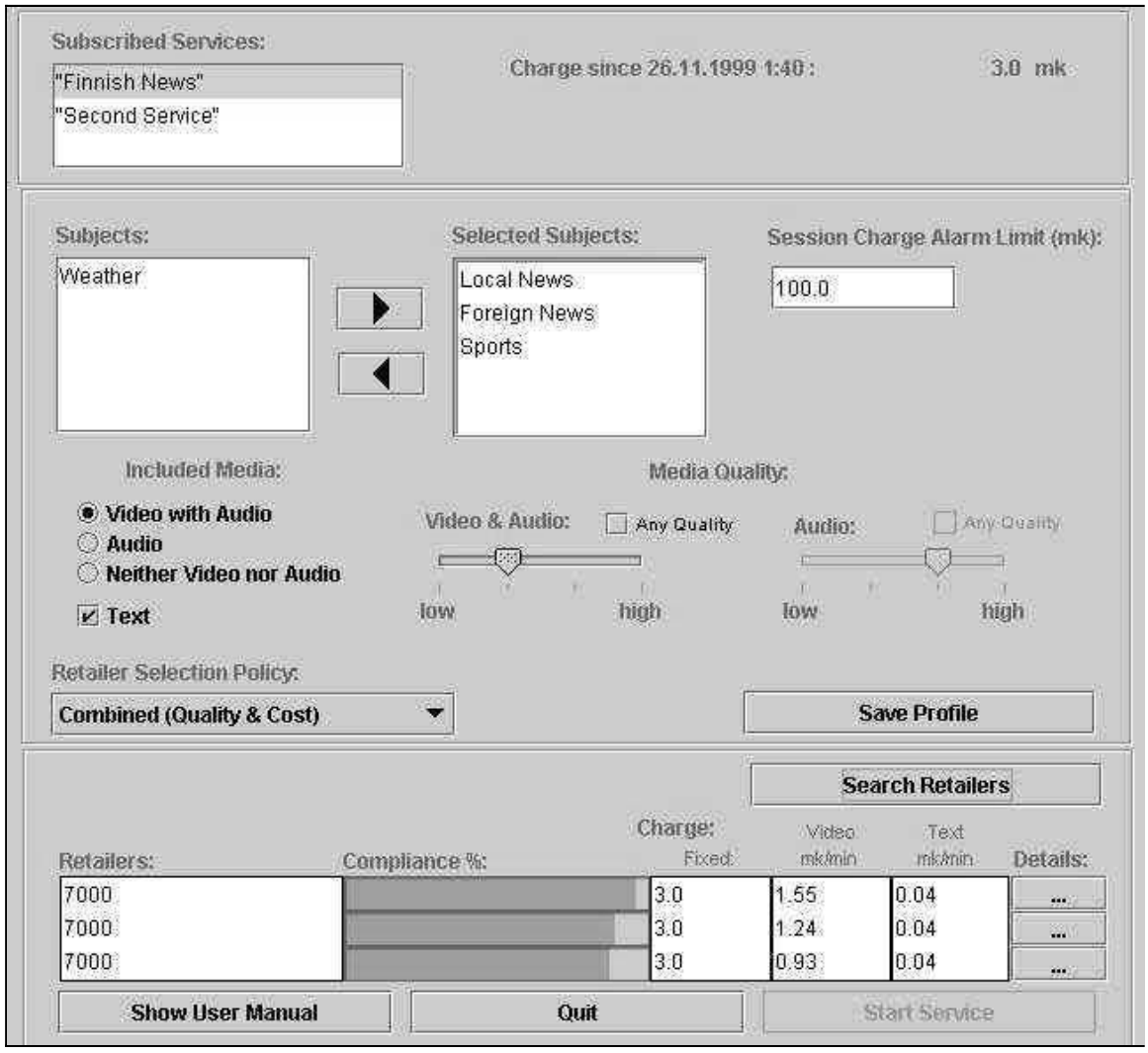


Fig. 2. Form for the Service Settings

From the service point of view the user interface applet may be seen as a form (term hereby used) for filling out the customisation information for desired service usage. The form (presented in Fig. 3) was planned together with usability experts, since it plays an important role in getting the correct feedback from the user and then helps the agents to learn the user preferences. The initial values are read from the persistent user profile and then adjusted according to the user's choices.

The form has been divided into three main rectangles. The first contains the list of services along with the cumulative charge from the beginning of current billing period. Under that is the rectangle for customising the news service usage. The service consists of sections that can either be deselected (the left-hand side) or selected (the right hand side). Next to those is a limit to the allowed charge for service usage to guide the negotiation procedure.

The service offers are differentiated by the number of media and the quality of service. This is reflected at the form by having radio buttons for selecting the media to be included. The choice between "video with audio" and "audio" is exclusive since video without audio was not regarded practical. The choice of text is however independent. Having chosen the included media the user may then adjust their quality by using the sliders.

The third rectangle is for commencing the negotiation (button "Search Retailers") and provides the space for presenting the results. The user gets as negotiation output the name of the service provider (during the development the port number of the Voyager server was used, therefore the number "7000"), the fixed costs of the service and the variable costs per each medium. The compliance bar in the middle reflects how well the offer matches the parameters specified by the user.

2.3 Content Filtering and Delivery

The service chosen for the trial was a news delivery service. The news material was digitised from the TV news of the Finnish Broadcasting Company, and further post-processed in order to get the different quality levels, audio only and text. All this material was stored to the file system of the content provider node, but the SP was responsible for the provision of the content and the metadata created by the content provider was reused by the SP for the creation of offers.

In order to activate a service the user chooses one of the three offers presented to him and presses the button "Start Service". This causes, quite naturally, the chosen service provider to take the role of the selected service provider. In order to provide the user with the service, the selected SP interacts with the content provider using agents in various roles. An overview of the different components is given in Fig. 3.

The User Agent Selected (UAS) is a mobile agent that migrates from the Visited SP to the selected SP at the beginning of the service session where it represents the user. Its main role is to act as component routing user requests to the content-provision-related components and to control the user requests for starting and stopping sessions. It locates a component called Service Factory (SF) which main task is to create for each user's session an entity called Service Session Manager.

The Service Session Manager (SSM) has the functionality and the responsibility to control the personalised service session. It collects meta-information from the Metadata

Agent and then asks the Service Specific Agent to filter out the URLs that do not match the users filtering profile. Next, the SSM asks the MA to generate (using the html-generator) a personalised index-page that contains only the selected content.

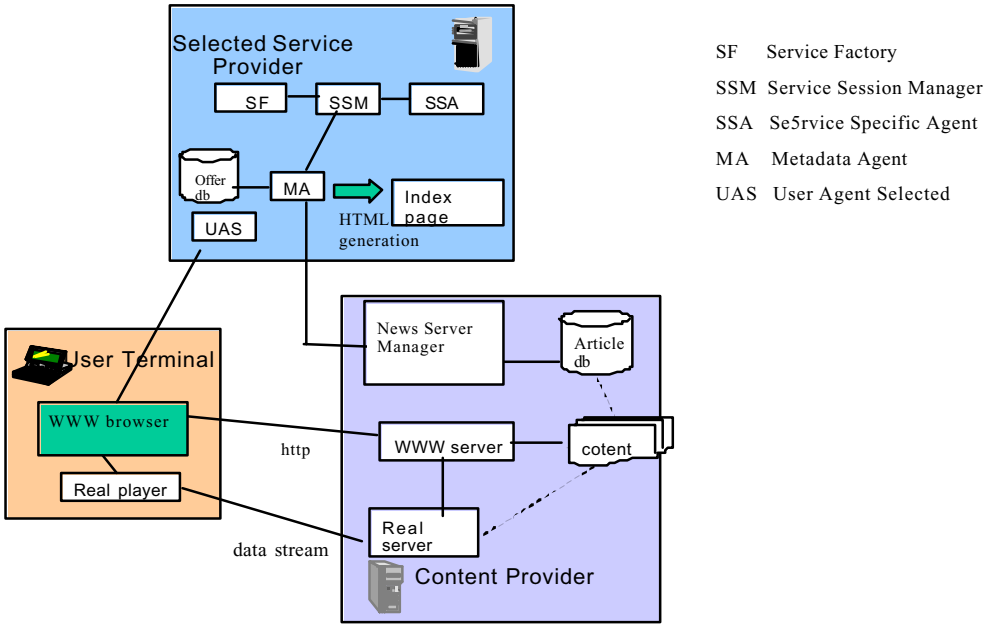


Fig. 3. Overview of the Service Provision

The Metadata Agent (MA) collects metadata from all available content providers and keeps the SP updated on available content and services. Additionally it is responsible for frequent updates of the current data.

When news stories are archived for a longer period of time, the actual amount of content may grow quite huge. The problem was addressed by creating a Service Specific Agent (SSA) that is able to assis t the user. The idea is that the user is presented only with the most current material, but using the SSA the user can also access the old one. In addition, the SSA has the capability of doing searches based on several criteria, the author and source of the material, keywords etc., that are reflected in the collected higher-level information.

The News Server Manager (NSM) is the counterpart of the MA in the content provider side. It collects the higher-level information of the content from a database and makes it accessible through a Voyager interface to the MA in the selected service provider. This way the possibly different means of database access (SQL in the example content) are abstracted away so that standardised agents can be used.

3 Field Trial

In order to evaluate the maturity of the mobile agent technology, a field trial was constructed and executed during November 1999. Also a usability test was arranged to catch potential design flaws concerning the user interface. The service described in the

former chapters was ported into an open pilot network, where it was advertised to attract as many users as possible. The exercise as a whole included crucial phases of the development and enrolment of telecommunication software and network services.

The trial was run in Mediapoli network located in Espoo, Finland. Mediapoli is a research and pilot network jointly initiated by Helsinki University of Technology and a number of research institutes. The network connects the research institutes with the households of the campus area as well as new media companies in a business park and forms a piloting community with more than 10 000 connected end users. Mediapoli offers a prototype of the future (year 2010) information infrastructure to be used for piloting of new services and network technologies. The provided network technology is switched Ethernet with gigabit links between the switches and 10 M bit/s access links to the households. Since the Mediapoli is targeted at piloting of new media, research installations and education purposes, it allows use of unconventional and new hardware and software technologies.

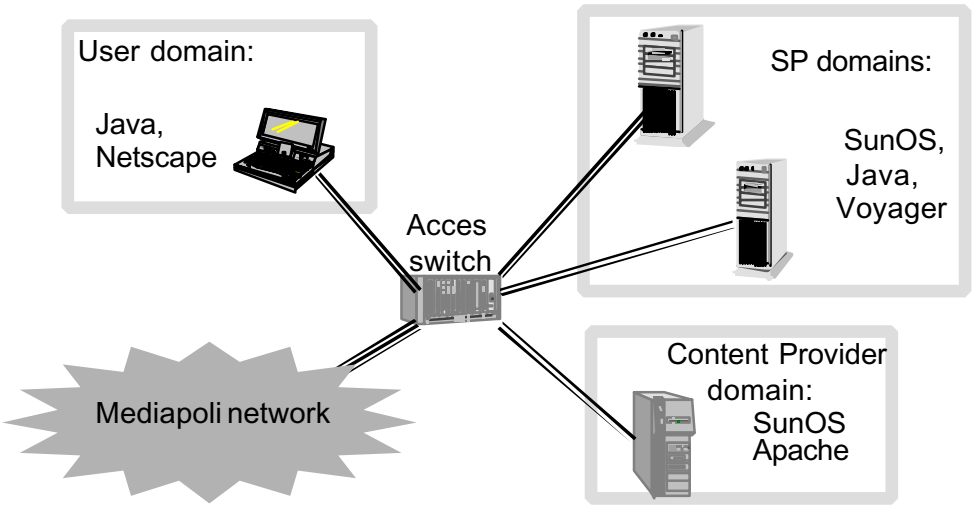


Fig. 4. Hardware Configuration constructed for the user trial

Fig. 4 depicts the trial hardware configuration. The service provider nodes as well as the workstation delivering the news material were located in a laboratory of VTT Information Technology. The content provider node hosted a Web server acting as a multimedia content and application provider. The test users were able to access the service from their home PC's or from the institutions connected to the pilot environment. A PC implemented a user domain with Internet connectivity and a Netscape Web browser as a minimum software requirement. The user terminal did not need to support mobile agents, since they were used only in the communication between service providers. Both SP signalling and service delivery relied on IP for connectivity.

The prototype implementation was made in Java with Voyager [7] as the agent platform. The software integrated the service access scenario, including user agent migration and service selection, with a database of metadata and the news server. The service was implemented with the following technology: SPARC workstation, Solaris 2.5.1, Java JDK 1.2, Voyager3.0.

The news material originated from the Finnish national broadcasting company Oy Yleisradio. Daily news broadcasts were captured, digitised and converted into the format of a commercial video server (RealServer). Quality of service variation was implemented by storing three copies of each video clip in low, medium and high bandwidth resolution. A database of metadata (such as keywords, news category, a link to the content page) was implemented and updated daily during the trial. An overview of the service provision architecture is shown in Fig. 4.

The field trial was executed by making the news delivery service publicly known to Mediapoli users and available in the network. Potential users could connect to the news delivery service, use the services freely and answer an on-line questionnaire on a volunteer basis. The challenge with this approach in general is the motivation of the users, e.g., to find a way to tempt them to use the service. The problem was dealt with a carrot approach: A number of movie tickets were given out by drawing a lot among the assigned users. The MONTAGE service installation and adoption were also made easy enough for the user, through the provision of installation instructions and a user manual that could be found in the field trial web pages [9]. The links referred to the following topics: (i) charging schemes, (ii) the selection process, (iii) a brief description of service usage scenarios, (iv) a user manual, (v) installation instructions and (vi) the on-line questionnaire in both English and Finnish. Also a help desk was provided offering email and phone support during the trial.

4 Evaluation

A major goal of the presented work was the assessment of the usefulness of agent technology and of its applicability in the envisaged context for all involved stakeholders. To achieve this goal such assessment criteria as ease of implementation, performance, service customisation, modularity, or integration ability with legacy applications have been used. The evaluation was based on data collected by means of: (1) a questionnaire for service developers, (2) on-line questionnaire for service users, (3) data logs on performance collected during trial and manual logging and observations reported from the trial. In the following text the feedback from developers and end users is first presented and discussed. In section 4.3 the performance of an agent implementation is compared with the performance of corresponding static object implementation. Additionally the experiences with the trial software support are discussed.

4.1 Feedback from the Developers

Persons from six companies were involved in the development stage of the project. Most of the developers had previous experience with CORBA, but not with agent technology. All of them were asked to fill a questionnaire that was created at the beginning of the project. They answered to the questionnaire twice – first time after the laboratory trial and second time after the final trial involving mobile users. The purpose of the developer questionnaire was to collect information about agents as an implementation technology. It contained questions about observations during the development and focused on the modularity, maintainability, interoperability and

performance of the used operating systems, programming languages, middleware and agent platforms. Additionally the developers were asked to estimate their professional experience and their role in the development phase of the project.

During the first stage of the project two platforms, OrbixWeb and Voyager 1.0.1, were used, but some other agents platforms were tried (e.g., Grasshopper). Almost all developers thought that both, agent technology and CORBA, are easy to use (58,3 % - easy, 25% - very easy), but they pointed out that the easiness depends on the platform used, that is, whether it can hide the problems in distributed programming or not. Regarding this point, for instance Voyager seemed much easier to use than Grasshopper.

Asked if agent technology helped to increase the modularity, 54% of the developers answered positively. The same question regarding a given platform (Voyager) gave a much better result: most of the all developers (83%) believe that Voyager does it. The opposite answers were given regarding the maintainability. In that case CORBA was considered more useful, as intelligence and mobility can make agents difficult to maintain. Additionally usage of the two platforms made tailoring some events more complicated what was thought to influence the performance. As having two platforms made also the integration unnecessarily complicated, soon after release of Voyager 3.0 the decision was made to use Voyager 3.0 instead of Voyager 1.0.1 and OrbixWeb.

The developers were also asked of their opinion on the impact of agent technology to the performance. More than half of them (66,7%) said that agent technology might increase the performance, but they were not sure. The included comments may imply that some of the developers confused two different concepts: programming in Java and using agents. This could be related to the fact that all the agent platforms tried during the project were written in Java.

The idea of repeating the questionnaire was to investigate how the long term implementation experience have changed the initial opinions of the implementers. It turned out that at the end of the projects, after gaining more experience, the developers were less convinced that CORBA, agent platforms in general and Voyager in particular helped much to increase modularity, maintainability, performance and interoperability. In other words, software developers had been more optimistic after the development of the first year. This actually shows the opinion of many developers also outside of the project. Being relatively easy to learn, agent technology allows fast involvement into the development phase of a project and accelerates the implementation, but it does not really solve all the problems from the developers point of view. However, it does make sense in specific situations as for instance in negotiations where there is heavy exchange of traffic. and accelerates

4.2 Feedback from the End Users

As in the case of the developers, also the users of the offered services were interviewed in two phases of the development, but this time the users' groups were not identical. After the first phase, a laboratory usability test was carried out by usability specialists interviewing the users. In this case, the concept of automated selection of service providers and the settings of the system were explained in detail to the users. The goals of the test were to identify the main usability problems in the service and study user's attitudes and feedback on the utility of the service, but also to get the users opinion

about their trust in agents making decision on the users' behalf. From the answers it followed that the users found the system efficient or quite efficient, and most of them agreed that the general concept of having an automated system of choosing a service provider is a good idea. At this stage the users stressed an importance of a good user interface which would help in making decisions. It was important, as on-line feedback was used by the agents itself, since one of the agents used user satisfaction as an input parameter in its learning algorithm. The agent was designed and equipped with intelligence to learn the user's preferences for a service and to negotiate with a number of service providers. The goal was to find, on behalf of the user, the service offering that matched the user preferences the best. As soon as the user modified his preferences on a service, the agent was "customised" accordingly to match the modified requirements.

The results of the laboratory trials were analysed, decisions for necessary changes were made and the changes were implemented before the final trial. The final trial was carried out with voluntary technology student as the test users. They used the service at their homes, with their own PCs connected to the Mediapoli network. According to the log files 61 access sessions and 74 service sessions occurred during the three-week test period. The users were asked to fill on-line questionnaire. The feedback from users was needed to collect end user comments on agent-based service, in particular to assess user trust in the agent-based service provider selection process. In an open user trial where the service could be freely accessed, the on-line questionnaire seemed to be the only channel to collect user responses on performance and reliability. The service installation and adoption were also made easy enough for the user, through the provision of installation instructions and a user manual that could be found in the field trial web pages. Also a help desk was provided offering email and phone support during the trial.

The results of the questionnaire showed that, as during the laboratory interview, the users liked the idea of some kind of an "agent" or mediator representing them in choosing the best offer from a set of service providers. They also found the system quite efficient in making for them the choice of a service provider that matched their settings. Over 60% trusted that the system can choose the best service provider, however, 50% of all the users indicated they would have rather choose the service provider themselves. Taking into account that the users were not financially harmed by incorrect choice, the acceptance of the automated selection was rather low. It can be perhaps explained by not enough information about which factors and in what way affected the choice. Among the users that preferred to make the decision themselves, almost all stated that it was unclear for them how changing settings affects the choice of service providers. This shows the importance of a user interface in gaining user's trust, in this case in the automated selection of service providers. This should be considered while developing agents that make selection on behalf of a user.

4.3 Performance Measurements

In order to evaluate communication cost of mobile agents, a comparison between mobile agent and static object implementation was conducted. A similar functionality was implemented with two different methods of communication: the traditional CORBA communication with remote procedure call, and agent communication via migration.

Data collected and analysed regarding the performances mainly concerned the generated traffic load.

The measurements were focused on two mobile agents – UAA and SUA (see 2.1). UAA was responsible for providing the remote user with his home profile and represented the user in the service provider domain throughout the user's access session. SUA was used for negotiation with several service providers in order to find the most optimal service offer on behalf of the user.

In case of UAA the mobile agent implementation was considered in two cases. In one case only the data was moved while the code was linked from local library. In the other case both, code and data were transferred with the agent to allow the user's home service provider for using its own negotiation algorithms. In the tests, all the service providers visited by the UAA and SUA were situated in different machines. The traffic between the workstations was recorded with a Sniffer protocol analyser. The focus was on data volume of the traffic (counted in bytes) and the results are presented below.

Table 1. Volume of TCP traffic generated in different implementations

	A) mobile agent with code and data	B) mobile agent with data	C)static object
UAA	68 484	14 347	1 849
SUA	not available	15 357	7 083

The results collected in case of the UAA showed that the volume of the mobile agent transferring code and data was about 70 Kbytes. The volume of the same agent transferring only data (user profile) was 14.5 Kbytes, while the static object (agent) using the traditional remote procedure call paradigm contained only 1.85 Kbytes. This shows that use of mobile agents transferring code should be used mostly when the implementation version of the agent behaviour is very relevant to their mission or it is justified by the lack of the specific code at the remote end. For instance, in the discussed case the limitation of UAA to sending only data (user profile) means that all the service providers have exactly the same model of the user profile (a standardised interface) and the user must be satisfied with the functionality offered by the visited service provider.

The difference in communication overhead between mobile agent and static implementation seems big. However, in the test set-up, the SUA migrated to another service provider node where it negotiated locally by exchanging three request/reply pairs. This is compared with a static SUA issuing three remote operation calls. Already if the negotiation algorithm would involve more than six request/reply pairs, the use of mobile agents would lead to a more efficient implementation than the use of remote communication from the communication economics point of view. This demonstrates the benefits of mobile code in case when a dialog between two entities involves an extensive exchange of data, as e.g. in the case of negotiations between a service provider and an agent representing a potential user of the provider's services. In the discussed case the performance of the SUA mobile agent negotiation was optimal from the user's point of view in the sense that SUA was replicated and all the replicas processed

simultaneously in different service provider nodes. The negotiation results only were passed back to the default service provider node.

A first look at the traffic measurement results gives rather pessimistic view of mobile agents, and the difference in communication overhead between mobile agent and static implementation seems big. However the difference may become insignificant when compared with the volumes of the application data (i.e., multimedia communication). Also, the overhead is not a big trade off when the availability of a preferred functionality needs to be assured. In the discussed example the SUA agent does not necessarily need to be available in foreign domain. In case if the user trusts more his home SP's algorithm for negotiation, SUA can be brought there within the UAA agent.

The functioning of the Client application was better with the Netscape browser than with the Internet Explorer. However, in both cases there were no actual crashes detected. Netscape was used during the development and this explains the small difference in the behaviour of the browsers.

5 Conclusions

The cost of investment when implementing a system is clearly influenced by the design and implementation methods used. Availability of well-suited design, implementation and testing tools have great effect on the efficiency of the system developer. Implementation and integration of the agent-based software should not be a significant overhead, for otherwise this would increase the cost of the approach. Issues of interest here are the length of the learning phase of the technology, the software integration time, the availability and efficiency of tools and supporting material. To make the implementation time as short as possible is exceptionally important in highly competitive environment as future Internet with its e-commerce, where implementation of new services will not take years, but months or weeks rather.

The results of the discussed project showed that agent technologies allow cost-effective implementation and ease of service customisation. Correctly chosen platform makes the technology easy to learn and use, which substantially reduce the time of implementation. Additionally, from the service implementers' point of view, using an active modelling paradigm can help to handle the increasing functional complexity involved with service creation and deployment. For instance, service selection specific functionality, like finding the most appropriate service offer in a distributed market based on a user profile, can be encapsulated in a mobile agent and sent to where services are actually offered by service providers. The service creation and deployment process can therefore focus on the service key aspects and thus stays separated from service deployment specific issues arising with distributed programming.

Beside the software engineering advantage of enhanced encapsulation and the clear separation between the key functionality and the aspects of deployment, mobile agent technology can offer further technical benefits. These potential technical advantages are reduced communication cost, reduced bandwidth usage, the possibility of using remote interfaces and the support for off-line computation. By distributing functionality, autonomous agents located at the right position in a network add another technical advantage, namely scalability. Much more nodes can be controlled without hitting

performance limits at the initiating node. These technical advantages can be noticeable by service providers and end users in terms of faster service access and reduced operational costs.

Another aspect is the users' trust and satisfaction. With well design user interface, where the users can understand how their demands are reflected in the choices made by the agents, when they can trust the agents, they rather prefer the agents to act on their behalf because, as one of the users commented, "*it's faster that way*".

However, agent technology brings also disadvantages. The communication overhead of mobile agents based communication in comparison to static implementations can be big. In order to avoid this, the mobile code should be used with consideration, e.g. in case when a dialog between two entities involves an extensive exchange of data as it is in negotiations between two entities. Another cases are when there is lack of specific code on the remote end, or finally when for competing reasons some code is preferred.

Acknowledgements

This work has been carried out within the framework of the ACTS MONTAGE project whose target was to develop, demonstrate and assess agent-based solutions to accounting, charging and personal mobility support in telecommunications environments. The authors wish to thank all colleagues involved in the project for the helpful discussions.

Reference

[1] Montage www page, http://montage.ccrle.nec.de/
[2] Vu AnhPham, Ahmed Karmouch, Mobile Software Agents: An Overview, IEEE Communications Magazine, July 1998.
[3] http://www.fokus.gmd.de/research/cc/ecco/climate/entry.html
[4] Andreas Kind, Mitiades E. Anagnostou, Malamati D. Louta, Jan Nicklisch, Juan Tous, "Towards the seamless integration of mobile agents into service creation practice", IS&N Conference proceedings, April 1999
[5] S. Albayrak et al., "Intelligent agents for telecommunications applications", Proceedings of IATA 98' Workshop, IOS Press, June 1998
[6] Nicholas R. Jennings, Katia Sycara, Michael Wooldridge, "A Roadmap of Agent Research and Development", Autonomous Agents and Multi-Agent Systems, 1, 7-38 1998
[7] http://www.objectspace.com/products/index.html
[8] http://www.mediapoli.com
[9] http://montage.ccrle.nec.de/Public/NewsDeliveryService/index.html
[10] H. Jormakka, K. Valtari, D. Prevedourou, A. Kind, K. Raatikainen, "Agent-based TINA Access Session Supporting Retailer Selection in Personal Mobility Context", Proceedings of TINA'99 Conference, April 1999