Mobile Agents are at the crossroads of two more ancient concepts: agent and mobility. The concept of agent appeared in the field of artificial intelligence (AI) in the late 1970s and is rather fuzzy, leading to many definitions. An agent is usually defined as a software servant that either relieves the user of routine, burdensome tasks such as appointment scheduling and e-mail disposition, or sorts the information that is relevant to the user’s current interests and needs. This definition has made ”agent ” a buzzword within both the academic and industrial worlds.

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. Even if they are defined as a special class of agents that have mobility as a secondary characteristic, it is more appropriate to consider mobile agents as the achievement of mobile abstractions (code, objects or processes). They are often considered as an alternative and/or a complement for other paradigms such as the well-established client-server. Instead of transferring large amounts of data between the client program and the server, a mobile agent moves to the host with the data and pertinent resources.

Mobile agents have been used in applications ranging from information retrieval to e-commerce, including telecommunications and network management. Although their proponents associate several benefits with their use, they remain a contentious issue because of, for instance, the lack of innovative applications backing their claims with concrete studies.

The aim of the workshop was to provide a unique opportunity for researchers, software and application developers, and computer network technologists to discuss new developments on the mobile agent technology and applications. The workshop focuses on mobile agent issues across the areas of network management, mobile applications, Nomadic computing, feature interactions, Internet applications, QoS management, policy-based management, interactive multimedia, tele-learning applications, and computer telephony integration.

August 2001

Samuel Pierre and Roch Glitho
Organizing Committee

General Chair:
Samuel Pierre
École Polytechnique de Montréal, Canada

Program Chair:
Roch Glitho
Ericsson Research Canada

Tutorial Co-Chairs:
Jean-Charles Bernard
École Polytechnique de Montréal, Canada
Wilfried G. Probst
Université du Québec à Montréal, Canada
Thomas M. Chen
Southern Methodist University, USA

Publicity Co-Chairs:
Ramiro Liscano
Mitel Corporation, Canada
Larbi Esmahi
National Research Council of Canada
Geng-Sheng Kuo
National Central University, Taiwan
Antonio Liotta
University of Surrey, UK

Local Management Committee
Jean-Charles Bernard, École Polytechnique de Montréal, Canada
Denis Monette, Ericsson Research Canada
Roch Glitho, Ericsson Research Canada
Sabine Kébreau, École Polytechnique de Montréal, Canada
Raymond Lévesque, Bureau des Congrès Universitaires, Canada
Program Committee

A. L. Ananda, NUS, Singapore
J.-C. Bernard, École Polytechnique de Montréal, Canada
A. Bieszczad, Lucent Technologies, USA
R. Boutaba, University of Waterloo, Canada
J. Delgado, Universitat Pompeu Fabra, Barcelona, Spain
P. Dini, Cisco Systems Inc., USA
L. Esmahi, NRC, Canada
B. Falchuk, Bellcore, USA
N. Fonseca, Unicamp, Brazil
V. Gay, UTS, Australia
M.-P. Gervais, LIP6, France
R. Glitho, Ericsson Research Canada
T. Gray, Mitel Corp, Canada
C. Grossner, Nortel Networks, Canada
H. H. Hoang, École Polytechnique de Montréal, Canada
E. Horlait, LIP6, France
J. G. Hulaas, University of Geneva, Switzerland
R. Impey, NRC, Canada
I. Jabbari, George Mason University, USA
P. Joye, University of Applied Sciences of Western Switzerland
A. Karmouch, University of Ottawa, Canada
R. Liscano, Mitel Corporation, Canada
T. Magedanz, IKV++GMD, Germany
C. Petitpierre, Swiss Federal Institute of Technology, Switzerland
A. Quintero, École Polytechnique de Montréal, Canada
S. Soulhi, Ericsson Canada
J.-F. Wagen, Univ. of Applied Sciences of Western Switzerland

MATA Workshop Steering Committee

Eric Horlait, LIP6, France
Roger Impey, NRC, Canada
Ahmed Karmouch, University of Ottawa, Canada
Thomas Magedanz, IKV++GMD, Germany
# Table of Contents

Dynamic Congestion Avoidance Using Multi-agent Systems .......................... 1  
*N. Rouhana, E. Horlait*

Use of Agents for Resolving Feature Interactions ................................. 11  
*A. Khoumsi, S. Cherkaoui*

IMAGO: A Prolog-Based System for Intelligent Mobile Agents .................. 21  
*X. Li*

Mobile Agents and Legacy Systems:  
How to Integrate Alternative Communication Paradigms .......................... 31  
*A. Puliafito, O. Tomarchio, L. Vita*

Agent-Based Distance Vector Routing ............................................... 41  
*K.A. Amin, J.T. Mayes, A.R. Mikler*

High Availability Connection Management via Software Bus  
and Mobile Agent in Network Management Environment .......................... 51  
*Y. Li, L. Meng*

Multipoint-to-Point Routing with QoS Guarantees Using Mobile Agents ........ 63  
*S. Gonzalez-Valenzuela, V.C.M. Leung, S.T. Vuong*

Dynamic Composition of Execution Environment  
for Adaptive Nomadic Applications .................................................... 73  
*S. Campadello*

Network Processing of Mobile Agents,  
by Mobile Agents, for Mobile Agents ................................................ 81  
*I. Satoh*

Design of a Mobile Agent-Based Workflow Management System ................ 93  
*Y.-H. Suh, H. Namgoong, J.-J. Yoo, D.-I. Lee*
<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Towards an Agent-Based Distributed Hierarchical Network Management System for All-Optical Networks</td>
<td>103</td>
</tr>
<tr>
<td>B. Khan, D.D. Kleiner, D. Talmage</td>
<td></td>
</tr>
<tr>
<td>A Policy Management System for Mobile Agent-Based Services</td>
<td>114</td>
</tr>
<tr>
<td>M. Lakhdissi, H. Harroud, A. Karmouch, C. Grossner</td>
<td></td>
</tr>
<tr>
<td>A Multi-agent Architecture for Intelligent Mobile Agents</td>
<td>124</td>
</tr>
<tr>
<td>S. Goutet, S. Pierre, R. Glitho</td>
<td></td>
</tr>
<tr>
<td>A Self-adaptable Agent System for Efficient Information Gathering</td>
<td>139</td>
</tr>
<tr>
<td>A. Liotta, G. Pavlou, G. Knight</td>
<td></td>
</tr>
<tr>
<td>An Agent-Based Approach to Full Interoperability and Allocation Transparency in Distributed File Systems</td>
<td>153</td>
</tr>
<tr>
<td>W. Fornaciari, V. Piuri, A. Prestileo, V. Zaccaria</td>
<td></td>
</tr>
<tr>
<td>A Proposal of PVCs Configuration Methodology Employing Mobile Agents</td>
<td>163</td>
</tr>
<tr>
<td>A.R. Cardoso, J. Celestino Jr</td>
<td></td>
</tr>
<tr>
<td>SIT A: Protecting Internet Trade Agents from Malicious Hosts</td>
<td>173</td>
</tr>
<tr>
<td>M. Perry, Q. Zhang</td>
<td></td>
</tr>
<tr>
<td>Applying Mobile Agents to Enable Dynamic, Context-Aware Interactions for Mobile Phone Users</td>
<td>184</td>
</tr>
<tr>
<td>M. Adler, J.M. Bradshaw, M. Mahan, N. Suri</td>
<td></td>
</tr>
<tr>
<td>Using Mobile Agents for Managing Control and Alarms in Urban Infrastructures</td>
<td>195</td>
</tr>
<tr>
<td>A. Quintero, D. Konaré, S. Pierre</td>
<td></td>
</tr>
<tr>
<td>The Spider Model of Agents</td>
<td>209</td>
</tr>
<tr>
<td>F.Y. Huang, D.B. Skillicorn</td>
<td></td>
</tr>
<tr>
<td>On the Modeling of Mobile Agent-Based Systems</td>
<td>219</td>
</tr>
<tr>
<td>F. Muscutariu, M.-P. Gervais</td>
<td></td>
</tr>
<tr>
<td>Multi-domain Policy Based Management Using Mobile Agents</td>
<td>235</td>
</tr>
<tr>
<td>N. Agoulmine, M. Fonseca, A. Marshall</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>A Mobile Agent Prototype for Distributed Search</td>
<td>245</td>
</tr>
<tr>
<td>E. Velazquez, N. Santoro, A. Nayak</td>
<td></td>
</tr>
<tr>
<td>Using the Cross-Entropy Method to Guide/Govern Mobile Agent’s Path</td>
<td>255</td>
</tr>
<tr>
<td>Finding in Networks</td>
<td></td>
</tr>
<tr>
<td>B.E. Helvik, O. Wittner</td>
<td></td>
</tr>
<tr>
<td>Context Gathering Mobile Agents to Assist</td>
<td>269</td>
</tr>
<tr>
<td>in Resource Reservation for Inter-technology Hand-off</td>
<td></td>
</tr>
<tr>
<td>S. Jamadagni</td>
<td></td>
</tr>
<tr>
<td>Towards a Secure Agent Platform Based on FIPA</td>
<td>277</td>
</tr>
<tr>
<td>M. Zhang, A. Karmouch, R. Impey</td>
<td></td>
</tr>
<tr>
<td><strong>Author Index</strong></td>
<td>291</td>
</tr>
</tbody>
</table>
Dynamic Congestion Avoidance Using Multi-Agents Systems

Nicolas Rouhana¹ and Eric Horlait²

¹ Faculty of Engineering, University Saint Joseph
P.O. Box : 11-514, Beirut, Lebanon
Nicolas.Rouhana@usj.edu.lb
² LIP6, University Pierre et Marie Curie
8, rue du Capitaine Scott, 75015 Paris, France
Eric.Horlait@lip6.fr

Abstract. In this paper, we present a mechanism called DWRED (Dynamic Weighted RED) that uses communicating agents in a Multi-Agents Systems to enhance the basic Random Early Detection (RED) congestion management algorithm. Our proposition provides different levels of service for multiple classes of traffic, and adds cooperation between the network nodes that allows dynamic modification of the running algorithm’s parameters in response to different network congestion states. We show preliminary results of a simple DWRED prototype implemented using the Multi-Agents Systems platform MADKIT.

1 Introduction

Congestion management is always a major concern in networks. Router queues fill during periods of congestion, and “last resort” congestion management is achieved through packet dropping. A popular congestion management technique that emerged is RED (Random Early Detection) [1] which controls average queue sizes, by deciding when and what packets to drop, using a random drop-probability factor when the average queue length is between a minimum and maximum threshold. But the major concern in RED is fine-tuning the algorithm’s parameters according to the network conditions in order to achieve appropriate results. Other more preventive mechanisms rely on notification messages (either backward or forward) to try to avoid packet drops [2] as well as relying on cooperation between network nodes and end hosts to respond to these notifications and slow down their rate to avoid reaching congestion. All these schemes call for engineering more “intelligence” inside the network, and result in having network entities interacting together to achieve the common goal of congestion management, which is closely related to “multi-agent” terminology.

“Agents” [3] are self-contained software elements containing some level of intelligence and responsible for performing part of a programmatic process acting on behalf of a user or an automated task. In general, the term “intelligent agent” ranges from
adaptive user interfaces, known as “interface agents”, to communities of “intelligent” processes that cooperate with each other to achieve a common task (“cooperative agents” or “multi-agents systems”). In this paper, we study the benefit of using Multi-Agents Systems (MAS) to control congestion in IP networks, by proposing a scheme called Dynamic Weighted Random Early Discard (DWRED) that extends RED by adding provision of service for multiple classes of traffic as well as dynamic modifications of the running RED parameters in response to network state. Hence, DWRED gateways modify the local RED queue management parameters in response to changing network conditions, and also notify their upstream neighbors to do the same accordingly through signaling messages. We show the results of DWRED by developing a prototype simulator using the Multi-Agents platform MadKit [4], which provides the means for communication between the network entities.

The next section describes our DWRED algorithm, section 3 introduces the Multi-Agents Systems concept and the MADKIT environment, section 4 presents the results of our prototype simulator, and conclusions are discussed in section 5.

2 Dynamic Weighted RED

RED gateways rely on several parameters to configure to which the congestion avoidance mechanism is rather sensitive to achieve acceptable results, otherwise behavior would be similar to the oscillations of the queue up to the maximum queue size found with Drop Tail gateways. The original RED packet drop probability is based on a minimum threshold $S_{\text{min}}$, a maximum threshold $S_{\text{max}}$, and a mark probability $\text{maxp}$:

- when the average queue $\text{avg}$ depth is above the minimum threshold, RED starts dropping packets randomly; if it exceeds $S_{\text{max}}$, all the packets are dropped. The computation of the average queue size $\text{avgt}$ at time $t$ uses a low-pass filter with time constant $\omega_q$ to smooth the variations of the current queue length $q$ and the value of the previously computed average $\text{avgt}_{t-1}$.

Furthermore, RED distributes losses in time and approximates a fair discard of packets among the flows without identifying them, i.e., assuming that all packets want to receive exactly the same service. The basic idea of our congestion avoidance scheme is to extend RED by two techniques. First, we combine the capabilities of the RED algorithm with IP Precedence [5] to provide Weighted RED (WRED) preferential traffic handling of priority packets. WRED selectively discards lower priority traffic and provide differentiated performance characteristics for different classes of service. WRED basically provides separate thresholds and weights for different IP precedences, allowing the provision different quality of service in regard to packet dropping for different traffic types. Second, we add control mechanisms, as explained below, that modify dynamically the drop preferences parameters of each class of traffic $i$ according to the average queue size.

In order to achieve our two goals, the output queue inside each router will be receiving the packets from all the classes, and for every class $i$, an initial minimum and maximum threshold $S_{\text{min}(i)}$ and $S_{\text{max}(i)}$ are defined. Fig. 1 shows the output queue and the different thresholds only for one class $i$ for the sake of clarity. In order to conserve
relative priorities, the class with the highest priority will naturally be chosen to have the highest thresholds. Of all the RED parameters, we chose to fix the maximum threshold to an initial value $S_{\text{max}(i)}$. The minimum threshold changes in a discrete manner according to pre-defined variations steps. The dynamic value of the minimum threshold at a given time for a class $i$ is given by $s_{\text{min}(i)}$, and varies from a minimum value $n_{\text{min}(i)}$ to the maximum value $S_{\text{min}(i)}$. Furthermore, and for the sake of simplicity, each of variables $w_q$, $\text{maxp}$, and the step of variation of $s_{\text{min}}$ are assigned a maximum and a minimum value depending on the congestion state, as shown in table 1. The local control entity assigns the appropriate value for each parameter according to the state of congestion in the network. The increasing and decreasing steps of $s_{\text{min}}$ (INC and DEC) can be either fast or slow (max or min) depending on the network congestion state.

Table 1. Various parameters values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Network state</th>
<th>No congestion (NC)</th>
<th>Heavy congestion (HC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_q$</td>
<td>$W_{q_{\text{min}}}$</td>
<td>$W_{q_{\text{max}}}$</td>
<td></td>
</tr>
<tr>
<td>$\text{maxp}$</td>
<td>$\text{maxp}_{\text{min}}$</td>
<td>$\text{maxp}_{\text{max}}$</td>
<td></td>
</tr>
<tr>
<td>$s_{\text{min}}$, INC step</td>
<td>$\text{INC}_{\text{max}}$</td>
<td>$\text{INC}_{\text{min}}$</td>
<td></td>
</tr>
<tr>
<td>$s_{\text{min}}$, DEC step</td>
<td>$\text{DEC}_{\text{min}}$</td>
<td>$\text{DEC}_{\text{max}}$</td>
<td></td>
</tr>
</tbody>
</table>

The dynamics of $s_{\text{min}}$ vary according to the state of the router indicated by $avg_t$:

- If $avg_t > s_{\text{min}}$ => $s_{\text{min}}$ is decremented by one step as long as it is above its minimal value $n_{\text{min}(i)}$
- If $avg_t < s_{\text{min}}$ => $s_{\text{min}}$ is incremented by one step as long as it is under its initial value $s_{\text{min}(i)}$.

For every arriving packet of class $i$, the average queue size $avg_t$ is computed over the total packets in the queue:

- If $avg_t < s_{\text{min}(i)}$ => no drop
- If $s_{\text{min}(i)} < avg_t < S_{\text{max}(i)}$ => random drop of the packet
- If $avg_t > S_{\text{max}(i)}$ => drop the packet

A more constraint way to further differentiate drop precedences would be to consider the total number of packets of class $i$ for computing $avg$ when a packet of the highest priority class $i$ arrives, then the total number of packets of class $i$ and class $i-1$ for computing $avg$ when a packet of the priority class $i-1$ arrives, and so forth. In our case, we always compute the average queue size based on the total number of packets.
in the queue, and we define an additional threshold \( n_{\text{min(i)}} \) maintained per class of traffic, under which no drop occurs as a way to guarantee a minimum amount of traffic for that class. So basically, drop actions taken for a class depend on the total packet count in that queue, with a minimum guaranteed of occupation in that queue.

### 2.1 Cooperative Dynamic Control

In order to further improve the congestion management, and in addition to locally controlling their parameters, routers respond to explicit neighbor solicitation to modify parameters. Three signaling messages defined are the following:

- **DEC (DECrement)** message sent by the local control entity to affluent routers when \( \text{avg}_t \) in the local queue is above \( S_{\text{max}} \) of a certain class. The routers receiving this message decrement \( s_{\text{min}} \) of the correspondent class if it is above its minimum value.

- **ADJ (ADJustment)** message sent by the local control entity to the affluent routers as well as to the local router when \( \text{avg}_t \) transits from a value below \( S_{\text{max}} \) of a certain class to a value above it. ADJ proposes to set the parameters to the HC state values.

- The message **RADJ (Re-ADJustment)** sent by the local control entity to the affluent routers as well as to the local router when \( \text{avg}_t \) transits from a value above \( s_{\text{min}} \) of a certain class to a value below it. RADJ proposes to set the parameters to the NC state values.

All these messages assure a cooperative reaction of the network for the congestion control as well as for providing a higher bandwidth when there are no congestion risks. The two messages DEC and ADJ allow effective and fast control reactions. Furthermore, the signaling messages between agents are kept short and concise so that they do not consume bandwidth; they are limited to \(<\text{message type, class ID}>\) in lieu of sending complete parameters, a router informs his neighbor what set of parameters to choose from. The messages are also exchanged when needed; there is no periodic polling.

Fig. 2 shows how the control entities change their parameters states upon reception of any of the three messages. The DWRED algorithm is implemented on every router, and contains all congestion control actions undertaken by the agents: for every packet arriving at the queue at time \( t \) determine the class of the packet, compute new average queue size \( \text{avg}_t \), and apply the following algorithm, which re-adjusts the service profile of the corresponding class as seen in Fig. 3:

![Fig. 2. States of the control entity](image1)

![Fig. 3. Resulting packet drop probability](image2)
if \( \text{avg}_t < S_{\text{min}(i)} \)
  \( \text{if } s_{\text{min}(i)} < S_{\text{min}(i)} \)
    increment \( s_{\text{min}(i)} \)
  \( \text{if } \text{avg}_{t-1} \geq s_{\text{min}(i)} \)
    send message \( \text{RADJ} \)
  \( \text{if } s_{\text{min}(i)} \leq \text{avg}_t \leq S_{\text{max}(i)} \)
    \( \text{if } s_{\text{min}(i)} > S_{\text{min}(i)} \)
      decrement \( s_{\text{min}(i)} \)
  \( \text{if } n_i > n_{\text{min}(i)} \)
    compute RED probability to reject packet
  \( \text{if } \text{avg}_t > S_{\text{max}(i)} \)
    \( \text{if } s_{\text{min}(i)} > S_{\text{min}(i)} \)
      decrement \( s_{\text{min}(i)} \)
    \( \text{if } n_i > n_{\text{min}(i)} \)
      reject packet
    send message \( \text{DEC} \)
  \( \text{if } \text{avg}_{t-1} \leq S_{\text{max}(i)} \)
    send message \( \text{ADJ} \)

In summary, the control entity acts on the local resources in an asynchronous and autonomous way, and its actions are controlled according to queue statistics. The entity is also capable to communicate directly with other control entities. It is capable of adaptation in a way allowing it to respond to the needs of his environment and other entities with whom it is interacting. It uses communication to cooperate with its environment to coordinate actions and collaborate for the resolution of congestion. The result is a group of entities interacting together to achieve the common goal of congestion management. So the dynamic control part is able to act intelligently; this attribute indicates that the method used for developing the “intelligence” is closely related to “agent” terminology, which is exposed in the next section.

3 Multi-Agents Systems

There are three basic types of agent-based service architecture: single-agents, multi-agents and mobile agents. In single-agents systems, agents can operate autonomously (they are often event or time triggered), and may communicate with the user, system resources as required to perform their task. In Multi-Agents Systems (MAS), more advanced agents may cooperate with other agents to carry out tasks beyond the capability of a single agent to achieve their individual goals. Finally, in Mobile Agents systems, as transportable or even active objects, agents may move from one system to another to access remote resources or to meet or cooperate with other agents. In this paper, we are interested by the Multi-Agent Systems approach, where the main concern is the coordination of intelligent behavior among a collection of autonomous intelligent agents, i.e. how do they coordinate their knowledge, goals, skills, and plans to jointly take actions or solve problems [6], which is a feature that is close to our congestion control intentions in the network (Fig. 4).
MAS-based agents are used in a wide range of applications, such as distributed vehicle monitoring, computer integrated manufacturing, natural language parsing, transportation planning, and in particular telecommunications management [7]. Agents in MAS (Fig. 5) are stationary entities in the network, providing the necessary intelligence, and are able to perform specific predefined tasks autonomously (on behalf of a user or an application).

The basic attributes of this type of agent are their ability to act asynchronously, to communicate, to cooperate with other agents, and to be dynamically configurable:

- **Asynchronous Operation**: An agent may execute its task(s) totally decoupled from its user or other agents. This means that agents may be triggered by the occurrence of a certain event, or by the time of day. An agent placed within the network may operate totally asynchronous to the user, performing its task by talking to various system resources and potentially to other agents.

- **Agent Communication**: During their operation, agents may communicate with various system resources and users. From an agent’s point of view, resources may be local or remote.

- **Agent Cooperation**: This attribute indicates that the agent system allows for cooperation between agent entities. This cooperation may necessitate the exchange of knowledge information, and represents the prerequisite for multi-agent systems.

The platform we chose to implement our control mechanism is MadKit (Multi Agent Development Kit) [4], which is a generic multi-agent platform written in Java, designed to support heterogeneous agent and communication models, and host multiple distributed applications. The MadKit kernel is a small agent engine that only manages the most basic functions in the platform: messaging, global structuration and agent lifecycle. It is completely decoupled from specific agent models and graphical user interface. Its small size and adaptability makes it compatible with the smallest devices (such as the Java Platform Micro Edition running on a Palm PDA). There is no “MadKit agent architecture”: the agent model is intentionally weak to ease integration of various classic agent models, while providing a group/role methodology for application design. An agent in Madkit is an active communicating entity that plays roles within groups. Groups are defined as sets of communicating and interacting agents. The role of an agent is an abstract definition of a function or service and is always defined within a group, and each agent can have many roles in his group(s) and can belong to several groups simultaneously.
4 DWRED Simulator

Our implementation of the DWRED simulator using MadKit is based on the sample network topology shown in Fig. 6. We consider three classes of traffic: Gold (high priority), Silver (medium priority) and Bronze (low priority).

Sources \(s_{10}\) and \(s_{11}\) generate gold traffic, \(s_{20}\) and \(s_{21}\) generate silver traffic, and \(s_{30}\) and \(s_{31}\) generate bronze traffic. R0, R1, and R2 are the routers implementing the agents executing DWRED buffer management. The entire routing and forwarding mechanisms of a router has been implemented by adding Java classes to Madkit to simulate a router behavior, because of the “generic” agents architecture in Madkit. In our case, all the agents integrated in the different routers are part of the same group, and their roles consist of:

- **Measurements** roles that monitor:
  - The instantaneous and maximal size of the queue
  - The number of packets of every class in the queue
  - \(s_{\text{min}}\) of every class, as well as other parameters of the network
- **Actions** defined as:
  - Variation of the \(s_{\text{min}}\) level
  - Random or total packet drops
  - R2 sending ADJ, RADJ and DEC messages to the upstream affluent routers R0 and R1

This example analyzes the case of a strong congestion in the network, as the incoming rate is higher that the outgoing line rates. Figures 7 and 8 show the evolution of the number of drops in all the routers of the network respectively in the static case (no modification done to the parameters) and the dynamic case (applying DWRED algorithm). The total percentage of loss is shown in table 2.

![Fig. 6. Simulated topology](image)

<table>
<thead>
<tr>
<th>Table 2. Traffic loss ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Static control</td>
</tr>
<tr>
<td>Dynamic control</td>
</tr>
</tbody>
</table>
We notice that the dynamic control allows a stronger resistance to losses; a bigger number of Bronze and Silver packets arrive to the destination. In the dynamic control, even the Gold packets have been penalized for the global profit. The priority between classes is also preserved. Furthermore, the monopolization of the link by one of the classes is less intense in the dynamic case, because of the fact that the agents assure a minimal level of service for every class in the queue, below which no packet of this class is rejected.

Fig. 7. Static control
Fig. 8. Dynamic control
5 Conclusions

Multi-Agents Systems (MAS) have already been applied to control congestion in ATM networks [2], and we adopted a similar approach in IP networks. Our simple DWRED simulator shows that cooperative congestion avoidance between routers could significantly improve service. Due to Multi-Agents Systems, the dynamics of the mechanism can be easily implemented and deployed, and communications between agents assure cooperation in the entire network to detect and fight any incipient congestion. The prototype implemented is a young but promising approach to adaptive congestion control according to the variable state and parameters of the network. DWRED is effective even in the absence of cooperation from the transport protocol, such as TCP. Furthermore, DWRED uses “backward” congestion notification which greatly reduces the control delay that feedback congestion control systems exhibit, as opposed to “forward” congestion notification schemes. Future works will include simulations that take into account cooperating end hosts to further study the improvement of the proposition in comparison to RED.

References

Use of agents for resolving feature interactions

Ahmed KHOUMSI and Soumaya CHERKAOUI

University of Sherbrooke, Dep. GEGI, Sherbrooke J1K2R1, CANADA,
(khoumsi,soumaya)@gel.usher.ca

Abstract. A Feature Interaction (FI) occurs when services (or features) behave incorrectly once they are used together. In this paper, we show how FIs can be resolved by using agents. An interest of our approach is that, instead of modifying directly the interacting services, we use a static or mobile agent which avoids the interaction by "forcing" the services to behave in a desirable way. We have determined a reduced set of generic operations that must be implemented and available to every agent; the possibility to execute these operations guarantees the possibility to resolve interactions using our approach.

Keywords: Feature interaction resolution, static and mobile agents.

1 Introduction

We say that a feature interaction (FI) [2] occurs when the joint use of two services (or features) induces an undesired behaviour. In [9] we proposed approaches for detecting and resolving interactions, and in [10] we proposed a detection method that has been applied to detect all the interactions in [7]. In the present paper, our objective is to show how the basic resolution principles determined in [9] can be realized by using static and mobile agents.

Among the different attributes used to describe agents, autonomy is the only attribute that is commonly agreed upon. Hence the simplest definition of an agent can be: an autonomous software entity. The most important aspects considered in the research on agents are: mobile agent technology (MAT) which is focused on mobility [5], and intelligent agent technology (IAT) which is focused on intelligence and co-operation [8]. In this paper, we have opted for a MAT-based resolution approach because, like in [1], we think that MAT is ready to use and provides more benefits in the field of telecommunications in the short-to-medium term time frame than IAT.

Being inspired by [3], we have determined two categories of interactions, depending on whether the two services involved in the interactions are implemented: (1) in the same component of the network or (2) in different components of the network. For both categories, an interaction is resolved by the use of a software agent (more simply an agent) which "forces" the two interacting services to behave in a desirable way. For the first category, the used agent is static, and for the second category the used agent is mobile and moves between the locations of the two services. An important contribution of our study is that we have determined a reduced set of generic operations which must be implemented.

© Springer-Verlag Berlin Heidelberg 2001
and available (as a library) to every agent. The possibility to execute these operations guarantees the possibility to realize the resolution principles of [9] by using a static or mobile agent, for any detected interaction. Another advantage is that we keep the interacting services “unmodified” and add to them an agent which “forces” them to behave in a desirable way.

As a MAT-based related work, [4] presents a complementary study that proposes a mixed architecture of generic static and mobile agents. Two types of generic agents are used: component agents (CA) and feature interaction agents (FIR). If we compare [4] and the present article, the latter proposes generic operations, while the former proposes generic agents.

The rest of this paper is structured as follows. In Sect. 2, we present two examples of interactions. In Sect. 3, we present a set of generic operations that guarantee the possibility to resolve interactions using our approach. In Sect. 4, we give several examples where FIs are resolved by using agents and the generic operations. And in Sect. 5, we discuss the contributions of this study and propose some future works.

2 Examples of feature interactions

Here are two examples of interactions, a centralized interaction and a distributed interaction, respectively. More examples will be presented in Sect. 4 to illustrate the proposed agent-based resolution approach.

A centralized interaction involves two services running in the same component of the network. The interaction considered here involves 911 and Three-Way Calling (3WC) services. The 911 service prevents anyone from putting a 911 operator on hold. The 3WC service allows a 3WC subscriber $A$ who is in communication with $X$ to put $X$ on hold by flashing the hook, and then $A$ can call $Y$; while $A$ and $Y$ are in a phone conversation and $X$ is on hold, $A$ can flash the hook a second time to add $X$ in the conversation. There is an interaction because 3WC cannot function correctly if $X$ is a 911 operator. In fact, the 3WC service has to put on hold a 911 operator who cannot be put on hold.

A distributed interaction involves two services running in different components of the network. The interaction considered here involves Operator Services (OS) and Originating Call Screening (OCS). Every subscriber can use the OS service which acts like an outgoing POTS call, except that it is operator-assisted. The OCS service allows to screen outgoing calls based on the destination number; more precisely, a OCS subscriber $A$ can put numbers in a screening list $L_{ocs}$, and then the service OCS blocks any attempt of $A$ to call a subscriber whose number is in $L_{ocs}$. There is an interaction because the intention of a OCS subscriber may not be respected. In fact, let us assume that $A : (1)$ is subscriber to OCS, (2) has put in $L_{ocs}$ the number of a subscriber $X$, and (3) tries to call $X$ by using OS. Since the switch of OS is different from the switch of OCS, therefore the OS operator cannot know the content of $L_{ocs}$, and thus, allows $A$ to call $X$.  

12 Ahmed Khoumsi and Soumaya Cherkaoui
3 Generic operations for resolving interactions

3.1 Principles

In [9] we proposed an approach for resolving interactions which is based on the following ideas, where S1 and S2 are two interacting services: (i) not to redesign S1 and S2, (ii) to assign a priority to a service relatively to the other service, (iii) to allow S1 and S2 to exchange information, (iv) to enable and disable services, (v) to intercept incoming and outgoing calls and events, and (vi) to interrupt incoming and outgoing calls. Point (i) is mandatory, while the other points are optional depending on the types of services and interactions. In this paper we show how FI resolution based on these ideas can be realized by using static and mobile agents. After a study of several interactions, most of them in [3, 6, 7], we determined a set of generic operations that are useful to the agents for resolving interactions. In order to guarantee that the resolution is possible, the generic operations must be: (1) available as a software library and (2) realizable, in each component hosting services.

3.2 Proposed set of generic operations

Each operation will be presented as a procedure which may receive input argument(s) and return output argument(s), that are prefixed by in and out respectively. When relevant, certain procedures are presented more than once, for different values of their arguments. In the definition of every procedure, which is executed in a given component C:

- “incoming event” means “event coming from the network and received in C”,
- “outgoing event” means “event sent in C towards the network”,
- “incoming call” means “callee part (in C) of a call process”,
- “outgoing call” means “caller part (in C) of a call process”,
- “filter an incoming event” means “hide an incoming event from its destination in C”,
- “intercept an outgoing call” means “hide an outgoing call from the network”,
- “intercept an incoming call” means “hide an incoming call from the callee”.

In the following, a variable representing an argument is in lower case and a specific value of an argument is in upper case. Here are now the generic procedures:

- \textit{Enable(in:service)} and \textit{Disable(in:service)}: These two functions are used to set service in a state from which it can (resp. cannot) be used. When service is disabled, it ignores every request addressed to it, except \textit{Enable(in:service)}.

- \textit{Filter(in:incoming-event)}: After the execution of this function, every incoming event is filtered if it is equal to \textit{incoming-event}. Therefore, the filtered event is not delivered to its destination.

- \textit{Filter(in:incoming-event, in:another-event)}: After the execution of this function, every incoming event is filtered if it is equal to \textit{incoming-event}. The filtered event is transformed into the \textit{another-event} before to be delivered to its destination.
- **NoFilter**\((\text{in:incoming-event})\) removes the effect of a previous \(\text{Filter(\text{in:incoming-event}) or Filter(\text{in:incoming-event, in:another-event}).}\)

- **Generate**\((\text{in:INCOMING-CALL, in:origin})\) generates the exact event(s) which correspond(s) to an incoming call from the (distant) \textit{origin}. This implies that the local component \(C\) sees an incoming call from \textit{origin}, while in reality \textit{origin} has not initiated any call.

- **Generate**\((\text{in:OUTGOING-CALL, in:destination})\) generates an outgoing normal call addressed to the (distant) \textit{destination}. The consequence will be the generation of an incoming call in the destination component. By “normal” we mean call generated by dialing the number of the destination.

- **Intercept**\((\text{in:OUTGOING-CALL, in:service, out:destination})\): This is a blocking function which intercepts the next outgoing call generated by \textit{service}. The function returns the number of \textit{destination}.

- **Intercept**\((\text{in:INCOMING-CALL, out:origin})\): This is a blocking function which intercepts the next incoming call. Therefore the incoming call is not sent to its local destination. The function returns the number of the \textit{origin}.

- **Interrupt**\((\text{in:INCOMING-CALL, in:point})\): After the execution of this function, every incoming call will be interrupted at a \textit{point} of the call process.

- **NoInterrupt**\((\text{in:INCOMING-CALL})\) removes the effect of a previous \text{Interrupt(\text{in:INCOMING-CALL, in:point}).}\)

- **Interrupt**\((\text{in:OUTGOING-CALL, in:point})\): After the execution of this function, every outgoing call will be interrupted at a \textit{point} of the call process.

- **NoInterrupt**\((\text{in:OUTGOING-CALL})\) removes the effect of a previous \text{Interrupt(\text{in:OUTGOING-CALL, in:point}).}\)

- **Resume**\((\text{in:point})\) resumes the last interrupted (incoming or outgoing) call, from a \textit{point} which may be different from the point where the call has been interrupted.

- **Write**\((\text{in:element, in:container})\) writes \textit{element} into \textit{container} (e.g., database).

- **Read**\((\text{in:container, out:content})\) reads the \textit{content} of \textit{container}. It returns \textit{content}.

- **Disconnect**\((\text{in:local-user})\) disconnects \textit{local-user} who is on the line.

- **IsInState**\((\text{in:local-user, in:state, out:boolean})\) returns a boolean value which indicates whether \textit{local-user} is in a given \textit{state} (e.g., busy).

- **IsSubscriberTo**\((\text{in:local-user, in:service, out:boolean})\) returns a boolean value which indicates whether \textit{local-user} is subscriber to \textit{service}.

- **GetInfo**\((\text{in:local-user, in:service, out:information})\) returns all the relevant information related to \textit{service} of \textit{local-user}.

- **SendMsg**\((\text{in:local-user, in:message})\) sends \textit{message} to \textit{local-user}.

- **IsInLoop**\((\text{out:boolean})\) is used by a mobile agent to know whether it has gone through a loop.

- **Move**\((\text{in:destination})\) allows a mobile agent to move to \textit{destination}.

- **Move**\((\text{in:content, in:destination})\) allows a mobile agent to move to \textit{destination}, carrying the information \textit{content} with it.
4 Examples of FI resolution using agents and generic operations

The operations proposed in Sect. 3 have been determined after a study of several interactions, most of them presented in [3, 6, 7]. We will illustrate the application of agents and operations for the resolution of eight interactions, where almost all the generic operations presented in Sect. 3 are used. The studied interactions are grouped in two categories: centralized interactions which involve services running in the same component, and distributed interactions which involve services running in different components. Henceforth a subscriber is said busy when he/she is on the line.

4.1 Resolution of centralized interactions

Each centralized interaction is resolved by using a static agent (SA) that is executed in the same component than the two services involved in the interaction.

911 and Three-Way Calling (3WC) (see also Sect. 2) 911 prevents anyone from putting a 911 operator on hold. 3WC allows a 3WC-subscriber \( A \) who is in communication with \( X \) to put \( X \) on hold by flushing the hook, and then \( A \) can call \( Y \); while \( A \) and \( Y \) are in a phone conversation and \( X \) is on hold, \( A \) can flush the hook a second time to add \( X \) in the conversation. There is an interaction because 3WC cannot function correctly if \( X \) is a 911 operator. In fact, the 3WC service has to put on hold a 911 operator who cannot be put on hold.

An approach of resolution is that when 911 is used, every attempt of the 3WC-subscriber to put the 911 operator on hold is not sent to 3WC. Therefore 3WC will not try to put the 911 operator on hold. This resolution is realized as follows, where FLASH denotes the event “Flashing the hook”. When 911 starts (i.e., when the called 911 operator picks up), an agent calls Filter\( (\text{in}:\text{FLASH}) \) and then every incoming FLASH will be filtered. When 911 terminates (i.e., when the called 911 operator hangs up), the agent calls NoFilter\( (\text{in}:\text{FLASH}) \) and then the event FLASH will no more be filtered.

Terminating Call Screening (TCS) and Automatic CallBack (ACB) TCS allows to screen incoming calls based on the originating number. More precisely, a TCS-subscriber \( A \) can put numbers in a screening list \( L_{tcs} \), and then TCS blocks any incoming call from a subscriber whose number is in \( L_{tcs} \). ACB automatically records the last incoming call of a ACB-subscriber \( A \) when the latter is on the line; let \( X \) be the caller of the recorded call; as soon as \( A \)'s line is free, a ACB-call to \( X \) is generated as follows: \( A \) receives an ACB-tone and when he picks up, \( X \) is automatically called. There is an interaction because the intention of the TCS-subscriber may not be respected. In fact, if \( A \) is subscriber to both TCS and ACB, and if \( X \) is in \( L_{tcs} \), then the attempt of \( X \) to call \( A \) succeeds from the moment when \( X \) is automatically called by the ACB-call.
An approach of resolution consists of intercepting the ACB-call and replacing it by an incoming call because: (1) TCS can block only incoming calls, and (2) the interaction is due to the fact the ACB-call is not blocked by TCS. This resolution is realized as follows. When a call is recorded by ACB of $A$, an agent calls $\text{Intercept(in:OUTGOING-CALL, in:ACB, out}$.$\cdot x)$ which intercepts the next ACB-call and then returns in $x$ the destination $X$ of the ACB-call, i.e., $X$ is the initiator of the call recorded by ACB. Then the agent calls $\text{Generate(in:INCOMING-CALL, in}$.$\cdot x)$ in order to generate an incoming call from $X$.

Call Forwarding (CF) and Originating Call Screening (OCS) CF allows a CF-subscriber $A$ to program an automatic redirection of his incoming calls towards another subscriber $X$. OCS allows to screen outgoing calls based on the destination number; more precisely, a OCS-subscriber $A$ can put numbers in a screening list $L_{ocs}$, and then OCS blocks any attempt of $A$ to call a subscriber whose number is in $L_{ocs}$. There is an interaction because the intention of a OCS-subscriber may not be respected. In fact, let us assume that: (1) $A$ is subscriber to both CF and OCS, (2) $X$ is in $L_{ocs}$, and (3) $A$ has programmed a redirection towards $X$. If $A$ calls his own number then the call is automatically forwarded to $X$. Therefore $A$ succeeds to call (indirectly) $X$ although the number of $X$ is in $L_{ocs}$. This case happens because the number of $X$ has not been dialed and OCS checks only dialed numbers.

An approach of resolution consists of intercepting the forwarded call (also called CF-call) and replacing it by a normal call because: (1) OCS can block only normal calls, and (2) the interaction is due to the fact the CF-call is not blocked by OCS. (By “normal call”, we mean a call initiated by dialing the number of the destination.) This resolution is realized by an infinite loop of the following two operations: (1) an agent calls $\text{Intercept(in:OUTGOING-CALL, in:CF, out}$.$\cdot x)$ which intercepts the next CF-call and then returns in the variable $x$ the destination $X$ of the CF-call, (i.e., the CF-subscriber has programmed a redirection towards $X$), and (2) the agent calls $\text{Generate(in:OUTGOING-CALL, in}$.$\cdot x)$ in order to generate a normal outgoing call to $X$.

Call Waiting (CW) and Personal Communication Services (PCS) CW allows a CW-subscriber $A$ to receive calls even when he is on the line. If $X$ calls $A$ who is in communication with $Y$, then $A$ is informed by a CW-tone. By flashing the hook, $A$ puts $Y$ on hold and is connected to $X$. Then $A$ may switch between $X$ and $Y$ by flashing the hook. PCS customers may be registered with the same CPE and they are not necessarily subscribers to the same services. There is an interaction in the following situation: (1) $A$ and $B$ are PCS customers registered with the same CPE, and (2) $A$ is subscriber to CW while $B$ is not. Let us assume that $B$ is on the line when somebody calls $A$. Since the line is busy, therefore the CW of $A$ is started. The consequence will be to interrupt $B$'s call.

A first approach of resolution, which gives the priority to $B$, consists of preventing $A$ from using CW when $B$ is on the line. This resolution is realized as
follows: as soon as $B$ is on the line, an agent calls $\text{Disable}(\text{in:}CW)$; and as soon as $B$ hangs up, the agent calls $\text{Enable}(\text{in:}CW)$. A second approach of resolution, which gives the priority to $\mathcal{A}$, consists of disconnecting $B$ as soon as the CW of $\mathcal{A}$ starts. This resolution is realized as follows: if $B$ is on the line, an agent calls $\text{Disconnect}(\text{in:B})$ as soon as CW starts. The agent can know whether $B$ is on the line by using $\text{IsInState}(\text{in:B, in:busy, out:x})$ which returns the answer in the boolean variable $x$.

4.2 Resolution of distributed interactions

Each distributed interaction is resolved by using a mobile agent (MA) which moves between the two components which contain the two series involved in the interaction.

Operator Services (OS) and Originating Call Screening (OCS) (see also Sect. 2) Every subscriber can use OS which acts like an outgoing POTS call, except that it is operator-assisted. OCS is introduced in Sect. 4.1. There is an interaction because the intention of a OCS-subscriber may not be respected. In fact let us assume that $\mathcal{A}$: (1) is subscriber to OCS, (2) has put in $L_{ocs}$ the number of a subscriber $\mathcal{X}$, and (3) tries to call $\mathcal{X}$ by using OS. Since the switch of OS is different from the switch of OCS, therefore the OS operator does not know the content of $L_{ocs}$ and, for this reason, allows $\mathcal{A}$ to call $\mathcal{X}$.

An approach of resolution consists of adding the content of the $L_{ocs}$ of $\mathcal{A}$ (denoted $L^A_{ocs}$) into the $L_{ocs}$ of the OS operator (denoted $L^OS_{ocs}$). We assume here that the OS operator is subscriber to OCS. This resolution is realized as follows. An agent in the component of OS calls $\text{Interrupt}(\text{in:INCOMING-CALL, in:BEFORE-CHECKING-LOCS})$, and therefore every incoming call will be interrupted at a point before $L^OS_{ocs}$ is checked. As soon as an incoming call from a user $\mathcal{A}$ is interrupted in point $\text{BEFORE-CHECKING-LOCS}$, the agent calls $\text{Move}(\text{in:A})$ and $\text{IsSubscriberTo}(\text{in:A, in:OCS, out:x})$, in order to move to the component of the caller $\mathcal{A}$ and check whether $\mathcal{A}$ is subscriber to OCS. If the returned $x$ of $\text{IsSubscriberTo}(\text{in:A, in:OCS, out:x})$ is False, then the agent calls $\text{Move}(\text{in:OS})$ and $\text{Resume}(\text{in:BEFORE-CHECKING-LOCS})$, in order to return to OS and resume the previously interrupted incoming-call. If on the contrary the returned $x$ is True, then the agent calls $\text{Read}(\text{in:L^A_{ocs, out:y}})$ (or $\text{GetInfo}(\text{in:A, in:OCS, out:y})$, $\text{Move}(\text{in:y, in:OS})$, $\text{Write}(\text{in:y, in:L^OS_{ocs}})$ and $\text{Resume}(\text{in:BEFORE-CHECKING-LOCS})$, in order to add the content of $L^A_{ocs}$ into $L^OS_{ocs}$ and resume the previously interrupted incoming call.

Originating Call Screening (OCS) and Customized Ringing (CR) OCS is introduced in Sect. 4.1. CR allows a CR-subscriber $\mathcal{A}$ to have several numbers associated with a single line. When a subscriber calls $\mathcal{A}$, the ringing tone received by $\mathcal{A}$ allows the latter to know which of his numbers has been dialed by the caller. There is an interaction because the intention of a OCS-subscriber $B$ may not be respected if $B$ has put in $L_{ocs}$ a number $N$ of a CR-subscriber $\mathcal{A}$ with the
intention to prevent outgoing calls towards $A$. A call from the phone of $B$ is not prevented if another number of $A$ is dialed.

An approach of resolution is the following: if a number $N$ of a CR-subscriber $A$ is added in the $L_{ocs}^B$ of a OCS-subscriber $B$ (denoted $L_{ocs}^B$), then the other number(s) of $A$ must also be added in $L_{ocs}^B$. This resolution is realized as follows. When a number $N$ of a subscriber $A$ is added in $L_{ocs}^B$, then an agent in the component of $B$ calls $\text{Move}(\text{in} : A)$ and $\text{IsSubscriberTo}(\text{in} : A, \text{in} : CR, \text{out} : x)$, in order to move to the component of $A$ and check whether $A$ is subscriber to CR. If the returned $x$ of $\text{IsSubscriberTo}(\text{in} : A, \text{in} : CR, \text{out} : x)$ is False, then the agent calls $\text{Move}(\text{in} : B)$ in order to return to the component of $B$. If on the contrary the returned $x$ is True, then the agent calls $\text{GetInfo}(\text{in} : B, \text{in} : CR, \text{out} : \text{numbers})$, $\text{Move}(\text{in} : \text{numbers}, \text{in} : B)$ and $\text{Write}(\text{in} : \text{numbers}, \text{in} : L_{ocs}^B)$, in order to add into $L_{ocs}^B$ all the numbers of $B$ ($N$ excepted because it is already in $L_{ocs}^B$).

Call Waiting (CW) and Automatic ReCall (ARC) CW is introduced in Sect. 4.1. In a way, the aim of a CW-subscriber is to be always seen as being free when he is called, even when he is on the line. ARC automatically records the last outgoing call of a ARC-subscriber $A$ if the called party $X$ is on the line. As soon as $X$’s line is free, a ARC-call to $X$ is generated as follows: $A$ receives an ARC-tone and when he picks up, $X$ is automatically called. There is an interaction because ARC of $A$ is not activated when $A$ calls a busy CW-subscriber, since the latter is always seen as being free. In other terms, CW has precedence on ARC.

An approach of resolution, which gives precedence to ARC, consists of disabling the CW of a CW-subscriber $B$ when the latter is called by a ARC-subscriber while he is busy. This resolution is realized as follows. As soon as the CW-subscriber $B$ becomes busy, an agent in the component of $B$ calls $\text{Interrupt}(\text{in} : INCOMING-CALL, \text{in} : BEFORE-STARTING-CW)$, and therefore every incoming call will be interrupted at a point before CW starts. As soon as an incoming call from a user $A$ is interrupted in point $BEFORE-STARTING-CW$, the agent calls $\text{Move}(\text{in} : A)$ and $\text{IsSubscriberTo}(\text{in} : A, \text{in} : ARC, \text{out} : x)$, in order to move to the component of the caller $A$ and check whether $A$ is subscriber to ARC. If the returned $x$ of $\text{IsSubscriberTo}(\text{in} : A, \text{in} : ARC, \text{out} : x)$ is False, then the agent calls $\text{Move}(\text{in} : B)$ and $\text{Resume}(\text{in} : BEFORE-STARTING-CW)$, in order to return to the component of $B$ and resume the previously interrupted incoming call. If on the contrary the returned $x$ is True, then the agent calls $\text{Move}(\text{in} : B)$, $\text{Disable}(\text{in} : CW)$ and $\text{Resume}(\text{in} : BEFORE-STARTING-CW)$, in order to return to the component of $B$, disable CW and resume the interrupted incoming call. Then the agent calls $\text{Enable}(\text{in} : CW)$, in order to enable CW for the next calls. When $B$ becomes free the agent calls $\text{NoInterrupt}(\text{in} : INCOMING-CALL)$, in order to stop the effect of $\text{Interrupt}(\text{in} : INCOMING-CALL, \text{in} : BEFORE-STARTING-CW)$ because the interruption must occur only when $B$ is busy.

Call Forwarding (CF) and Call Forwarding (CF) CF is introduced in Sect. 4.1. There is an interaction because an infinite loop may happen in the
following situation: (1) \( \mathcal{A} \) and \( \mathcal{B} \) are CF subscribers, (2) \( \mathcal{A} \) has programmed a redirection towards \( \mathcal{B} \), (3) \( \mathcal{B} \) has programmed a redirection towards \( \mathcal{A} \), and (4) \( \mathcal{A} \) calls \( \mathcal{B} \). This situation induces an infinite loop \( \mathcal{A}-\mathcal{B}-\mathcal{A}-\mathcal{B} \cdots \). More generally, we may have an infinite loop between more than two CF-subscribers.

An approach of resolution consists of checking the existence of a loop when a user tries to initiate a call. If such a loop exists, then the call is not initiated and the user receives a message informing him about the problem. This resolution is realized as follows. An agent in the component of every subscriber calls \textit{Interrupt(in::OUTGOING-CALL,in::WHEN-DESTINATION-KNOWN)}, and therefore every outgoing call will be interrupted at a point after the destination is known and before the call is sent. After the interruption of a call initiated by \( \mathcal{X} \) and addressed to \( \mathcal{Y}_1 \), the agent in the component of \( \mathcal{X} \) calls \textit{Move(in::\mathcal{Y}_1)} in order to move to the component of \( \mathcal{Y}_1 \). Then the agent calls \textit{IsSubscriberTo(in::\mathcal{Y}_1,in::CF,out::x)} in order to check whether \( \mathcal{Y}_1 \) is subscriber to CF and has programmed a redirection. If the returned \( x \) is True, then the agent calls \textit{GetInfo(in::\mathcal{Y}_1,in::CF,out::\mathcal{Y}(i+1))} in order to know the destination \( \mathcal{Y}(i+1) \) of redirection. Then the agent calls \textit{Move(in::\mathcal{Y}(i+1))} in order to go through several components of \( \mathcal{Y}_1, \mathcal{Y}_2, \ldots \). More precisely, after each arrival in a component of \( \mathcal{Y}_i \), the agent will have to execute the following procedure:

1. Call \textit{IsInLoop(out::x)}
2. If \( x \) is False Then:
   3. Call \textit{IsSubscriberTo(in::\mathcal{Y}_i,in::CF,out::y)}
   4. If \( y \) is True Then:
      5. Call \textit{GetInfo(in::\mathcal{Y}_i,in::CF,out::\mathcal{Y}(i+1))}
      6. Call \textit{Move(in::\mathcal{Y}(i+1))}
   7. Else (i.e., \( y \) is False):
      8. Call \textit{Move(in::X)}
   9. Call \textit{Resume(in::WHEN-DESTINATION-KNOWN)}
EndIf
11. Else (i.e., \( x \) is True):
12. Call \textit{Move(in::X)}
13. Call \textit{SendMessage(in::X,in::INFINITE-LOOP-DUE-TO-CALL-FORWARD)}
14. Call \textit{Resume(in::AFTER-PICKS-UP)}
15. EndIf

Here are some explanations of the above procedure. In \textit{Line 1}, the agent checks whether it has gone through a loop. \textit{Lines 2-10} correspond to the case where the agent has \textit{not} gone through a loop. In \textit{Line 3}, the agent checks whether \( \mathcal{Y}_i \) is subscriber to CF and has programmed a redirection. \textit{Lines 4-6} correspond to the case where \( \mathcal{Y}_i \) is subscriber to CF and has programmed a redirection. In this case, the agent determines the destination of redirection \( \mathcal{Y}(i+1) \) and then moves to the component of \( \mathcal{Y}(i+1) \). \textit{Lines 7-9} correspond to the case where \( \mathcal{Y}_i \) has not programmed a redirection. In this case, the agent returns to the initiator \( \mathcal{X} \) of the call and resumes the call from the point where it has been interrupted. \textit{Lines 11-15} correspond to the case where the agent has gone through a loop. In this case, the agent returns to the initiator \( \mathcal{X} \) of the call, sends a message to \( \mathcal{X} \) to inform it about the problem, and sets the call process as if \( \mathcal{X} \) has just picked up.
5 Conclusion and future work

In this article, we propose an agent-based method for resolving feature interactions. Our main contributions can be summarized as follows:

1. Every interaction is resolved without redesigning the two involved services.
2. Each interaction is resolved by using simple (static or mobile) agents.
3. We have determined a set of generic operations, whose availability and realizability in each component hosting a service, guarantee the possibility to resolve feature interactions using our approach.

In the near future, we intend to investigate the following issues:

1. To develop a framework which combines and extend two complementary studies, namely [4] and the present article. The latter proposes generic operations, while the former proposes generic agents.
2. To adapt our approach for specific architectures, for example Intelligent Networks and Internet Telephony.
3. To extend our approach for the resolution of interactions involving more than two services.

References

IMAGO: A Prolog-based System for Intelligent Mobile Agents

Xining Li

Department of Computing and Information Science
University of Guelph, Canada
xli@snowhite.cis.uoguelph.ca

Abstract. This paper presents the preliminary design of the IMAGO project. This project consists of two major parts: the IMAGO Application Programming Interface (API) - an agent development kit based on Prolog, and the MVM - a multithreading agent server framework. We focus on the IMAGO API and its communication model - a novel mechanism to automatically track down agents and deliver messages in a dynamic, changing world. Examples are given to show the expressive power and simplicity of the programming interface as well as possible applications of the proposed system.

1 Introduction

Mobile Agents are mainly intended to be used for network computing - applications distributed over large scale computer networks. In general, a mobile agent is a self-contained process that can autonomously migrate from host to host in order to perform its task on behalf of a (human) user. Numerous Mobile Agents systems have been implemented or are currently under development. System-level issues and language-level requirements that arise in the design of Mobile Agents systems are well discussed in [1].

Most of the Mobile Agents systems are based on scripting or interpreted programming languages that offer portable virtual machines for executing agent code, as well as a controlled execution environment featuring a security mechanism that restricts access to the host’s private resources. Some Mobile Agents systems are based on Java [2] [3] [4] [5] [6], and some are based on other object oriented programming languages or scripting languages [7] [8] [9]. As the primary identifying characteristic of a mobile agent is its ability to migrate from host to host, support for agent mobility is a fundamental requirement of a Mobile Agents system. An agent is normally composed of three parts: code, execution thread (stack), and data (heap). All these parts move with the agent whenever it moves. However, most of the Mobile Agents systems (especially those built on top of Java) only support weak migration - an agent moves with its code and data without its stack of the execution thread. Thus, the agent has to direct the control flow appropriately when its state is restored at the destination. For example, a Java-based agent captures/restores its execution state through the
Java's serialising/de-serialising feature which provides a means for translating a graph of objects into a byte-stream and thus achieves migration at a coarse granularity. This implies that an agent restarts execution from the beginning each time it moves to another host. As a result, the agent has to include some tracing code in order to find its continuation point upon each migration.

Another mobile agent framework which embeds a logic programming component is pioneered by Distributed Oz [11] - a multi-paradigm language (functional, logic, object-oriented, and constraint), and Jinni [10] - a lightweight, multi-threaded, Prolog-based language (supporting mobile agents through a combination of Java and Prolog components). Distributed Oz does not support thread-level mobility, instead, it provides protocols to implement mobility control for objects. In a user program, the mobility of an object must be well-defined under the illusion of a single network-wide address space for all entities (include threads, objects and procedures). Jinni implements computation mobility by capturing continuations (describing future computations to be performed at a given point) at the thread-level. A live thread will migrate from Jinni to a faster remote BinProlog engine, do some CPU intensive work and then come back with the results.

This paper will discuss the design of the IMAGO project. The origin of the word *imago* means that

An insect in its final, adult sexually mature, and typically winged state, or an idealized mental image of another person or the self.

*WEBSTER's Dictionary*

In my proposal, imagoes are programs written in a variant of Prolog that can fly from one host on the Internet to another. That is, an imago is characterized as an entity which is mature (autonomous and self-contained), has wings (mobility), and bears the mental image of the programmer (intelligent agent). From computer terminology point of view, the term IMAGO is an abbreviation which stands for Intelligent Mobile Agents Gliding On-line.

The IMAGO project consists of two major parts: the IMAGO Application Programming Interface (API) - an agent development kit based on Prolog, and the MLVM - a multithreading agent server framework based on a sequential logic virtual machine LVM [12].

The IMAGO API consists of a set of primitives that allows programmer to create mobile agent applications. In general, a mobile agents system provides primitives for agent management (creation, dispatching, migration), agent communication/synchronization, agent monitoring (query, recall, termination), etc. In most concurrent programming languages, communication primitives take the form of message passing, remote procedure calls, or blackboard-based. For example, SICStus MT [13] uses the asynchronous message-passing mechanism whereas Bin-Prolog [10] adopts the blackboard-based model. On the other hand, IMAGO explores a novel model: instead of passing messages among agents through send/receive primitives, the IMAGO implements agent communication through *messengers* - special mobile agents dedicated to deliver messages on the network.
The goal of MLVM is to present a logic-based framework in the design space of Intelligent Mobile Agents server. To achieve this, we need to extended the LVM to cope with new issues, such as explicit concurrency, code autonomy, communication/synchronization and computation mobility. In designing the MLVM, some practical issues, such as multithreading, garbage collection, code migration, communication mechanism, etc, have been specified, whereas some other issues, such as security, services, etc, will be investigated further.

2 Overview of Imagoes

The IMAGO system is an infrastructure that implements the agent paradigm. An IMAGO server resides at a host machine intending to host imagoes and provide a protected imago execution environment. An IMAGO server consists of three components: a network daemon to accept incoming imagoes, a security manager to deal with privacy, physical access restrictions, application availability, network confidentiality, content integrity, and access policy, and a MLVM engine to schedule and execute imago threads.

Generally speaking, an imago is composed of three parts: its identifier which is unique to distinguish with others, its code which corresponds to a certain algorithm, its execution thread which is maintained by a single memory block (a merged stack/heap with automatic garbage collection)[12].

There are three kinds of imagoes: stationary imago, worker imago, and messenger imago. An agent application starts from a stationary imago. It looks like that the wings of a stationary imago have degenerated, so that it has lost its mobility. In other words, a stationary imago always executes on the host where it begins execution. However, a stationary imago has the privileges to access resources of its host machine, such as I/O, files, GUI manager, etc. A stationary imago can create worker or messenger imagoes, but it cannot clone itself. There is only one stationary imago in an application. We can find the similarity that there is only one queen in a colony of bees.

Worker imagoes are created by the stationary imago of an application. A worker imago is able to move such that it looks like a worker bee flying from place to place. A worker imago can clone itself. A cloned worker imago is an identical copy of the original imago but with a different identifier. A worker imago can not create other worker imagoes, however, it may launch messenger imagoes (system built-in imagoes) to deliver messages. When a worker imago moves from one host to another, it continues its execution on the destination host at the instruction which immediately follows the invocation of the move primitive. As mobile agents are a potential threat to harm the remote hosts that they are visiting, the IMAGO system enforces a tight access control on worker imagoes: they have no right to access any kind of system resources except the legal services provided by the server. A messenger queue is associated with each worker imago which holds all attached messenger imagoes waiting to deliver messages.
Messenger images are agents dedicated to deliver messages. The reason of introducing such special purpose images is that the peer to peer communication mechanism in traditional concurrent (distributed) programming languages does not fit the paradigm of mobile agents. This is because mobile agents are autonomous - they may decide where to go based on their own will or the information they have gathered. Most mobile agents systems either do not provide the ability of automatically tracing moving agents, or try to avoid discussing this issue. For example, Aglet API does not support agent tracking, instead, it leaves this problem to applications. On the other hand, the IMAGO system allows messenger images to track worker images and therefore achieves reliable message delivery. The system provides several builtin messenger images. Programmer designed messenger images are possible but this kind of images can only be created by the stationary image. A messenger image is anonymous so that there is no way to track a messenger. However, it can move or even clone itself if necessary.

3 Imago API

The code of an image is enclosed in a pair of directives. Here we follow the Prolog convention such that a directive specifies properties of the procedure defined in Prolog text. Three pairs of directives are used for image definitions, and they share the same syntax. For example, the following code gives a syntactical pattern of a messenger image:

```prolog
:- begin messenger
  my_messenger(Receiver, Msg) ;:-
  messenger_body. ...
:- end messenger
```

In each image, one and only one clause is defined by ;:- which indicates the starting entry of the image, and the rest clauses, if any, are defined by the Prolog convention. The entry clause can not be explicitly called, instead, the IMAGO runtime system automatically provides a goal toward the entry clause after an image text has been prepared for execution.

Even though several image definitions can be placed in a single source file, the IMAGO compiler will compile them independently and save the bytecode of each image into a separate file (the file name is composed by the name of its entry clause with a postfix .ima). For the above code pattern, its bytecode file is named as my_messenger.ima.

Messenger images are anonymous. As there is only one stationary image in an application, we reserve a special name queen for it. Names of worker images must be presented at the time they are created.

Like other logic programming systems, the IMAGO API is presented as a set of builtin predicates. This set consists of builtin predicates common to most Prolog-based systems and new builtin predicates extended for mobile agent applications. As we mentioned before, resource access predicates and user-machine
interface predicates can be used only in a stationary imago. In addition, the usage of agent management predicates depends on the type of imagoes, such as illustrated in Table 1 which lists predicates legal to each imago type. This table is far from complete, but should be sufficient to describe my project proposal.

<table>
<thead>
<tr>
<th>Imago Type</th>
<th>Builtin Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>stationary imago</td>
<td>create, accept, wait_accept, dispatch, terminate</td>
</tr>
<tr>
<td>worker imago</td>
<td>move, clone, back, accept, wait_accept, dispatch, dispose</td>
</tr>
<tr>
<td>messenger imago</td>
<td>move, clone, back, attach, dispose</td>
</tr>
</tbody>
</table>

Table 1: Builtin Predicates for Imagoes

In principle, all these predicates are not re-executable. Different kinds of errors, such as type error, resource error, system error, etc., might happen during their execution. However, for the sake of simplicity, we discuss these predicates in an informal manner, i.e., we only present a brief procedural description for each predicate.

**create(Worker_file, Name, Argument):** Create is used only by the stationary imago. It will load the Worker_file, spawn a new thread to execute the worker imago, put this new thread into the ready queue, and set up the imago’s Name and initial Argument.

**dispatch(Messenger_file, Receiver, Msg):** Dispatch is used to create a messenger imago which is responsible to search for the Receiver and deliver Msg. A worker imago can only dispatch system builtin messengers (which will be automatically created by imago servers), whereas the stationary imago can dispatch either system builtin messengers or programmer designed messengers (which can be loaded from the local file system). A messenger will implicitly carry the sender’s name (name of the imago which invokes the messenger) which is accessible by some other predicates.

**attach(Receiver, Msg, Result):** Attach is used only by messenger imagoes and probably the most complicated predicate in the IMAGO API. It will first search for the Receiver through its server’s log or probability through the IMAGO name server, instantiate Result to moved(S) if the receiver has moved to another host S, or deceased if the receiver could not be found. On the other hand, if the receiver is found currently alive, it will deactivate the calling messenger, and attach the caller to the receiver’s messenger queue. As soon as a messenger has been attached to the receiving imago, its thread is suspended until the receiver executes certain predicate to resume its execution. In this case, we say that the attach predicate is blocked.

**move(Server):** Invoking move allows a worker or a messenger to migrate to another imago server. This predicate deactivates the caller, captures its state, and transmits it to the given remote Server. When a worker issues move and there are pending messengers in its messenger queue, all these suspended messengers will be resumed and the term moved(Server) will be instantiated to the Result of each blocked attach predicate. This does not apply to a moving messenger, because messengers are anonymous and thus there is no way to attach a mes-
senger to another messenger. However, a resumed messenger should follow the moving worker to the new host in order to deliver its message.

**clone**\((\text{Name}, \text{Result})\): Clone will duplicate the caller (either a worker or a messenger) as a new imago thread with the given \text{Name} (anonymous for a messenger). The behavior of \text{clone} resembles the \text{fork()} in C where two imagoes continue their execution at the instruction immediately following the \text{clone} predicate but each has a different \text{Result} instantiation: \text{origin} to the caller imago and \text{clone} to the duplicated imago. When a worker issues \text{clone} and there are pending messengers in its messenger queue, all these suspended messengers will be resumed and the term \text{clone}(\text{Clone}) will be instantiated to the \text{Result} of each blocked \text{attach} predicate. Under this case, a resumed messenger must clone itself and then the original messenger re-attaches itself to the original receiver and the cloned messenger attaches itself to the cloned worker imago. A messenger example can be found in next section.

**back**: An imago calling \text{back} will move itself back to the host where the stationary imago resides in. The same as the \text{move}, this predicate will resume all pending messengers of a worker and bind \text{Result} to \text{moved}(\text{stationary_server}). Thus a resumed messenger should follow the receiver back to their home station.

**accept**\((\text{Sender}, \text{Msg})\): Stationary and worker imagoes can issue an \text{accept} to receive a message. It will succeed if a matching messenger has been found and the messenger will be resumed with an instantiation \text{received} to the \text{Result} argument, or it will fail if either the messenger queue is empty or no matching messenger can be found. \text{Accept} will never block, and is powerful enough to achieve indeterministic message receiving.

**wait_accept**\((\text{Sender}, \text{Msg})\): \text{Wait_accept} will cause its caller to be blocked (from the ready queue to a waiting queue) if either the caller's messenger queue is empty, or no matching messenger is found. It will succeed immediately if there is a pending matching messenger. An imago being blocked by this predicate will become ready when a new messenger attaches to it. A resumed imago will automatically redo this predicate; it succeeds if the new attached messenger matches, or it blocks the imago again otherwise. In other words, a \text{wait_accept} will never fail. It either succeeds or becomes blocked waiting for a matching messenger.

**dispose**: \text{Dispose} terminates the calling imago. All the pending messengers, if any, will be resumed with a \text{Result} bound to \text{deceased}. It is up to messengers to determine if they also dispose themselves or move back to notifying their senders.

**terminate**: This predicate is called by the stationary imago to terminate the application and eliminate all imagoes spawned (cloned) from this application.

A messenger attached to a worker imago is ready to be searched by \text{accept} or \text{wait_accept}. The behavior of an accepting predicate is determined by the unification of its arguments against pending messengers: it succeeds if a matching messenger is found, or it fails/waits otherwise. A failed accepting predicate does not cause any side effect and the messenger queue remains unchanged.
From the state transition description, we can find that a stationary or a worker imagos becomes blocked only if its messenger queue is empty or no matching messenger is found at the time a `wait_accept` is invoked, and it is resumed to ready when a messenger attachment occurs. On the other hand, a messenger becomes blocked when it is attached to a receiver, and it is unblocked when its receiver evaluates one of the following predicates: `move`, `clone`, `back`, `dispose`, or `accept` if the messenger matches.

4 Messenger Imago

The IMAGO system provides a set of builtin messenger imagos as a part of the IMAGO API. These messengers should be robust and sufficient for most imagos applications. They may be dispached by either a stationary imagos or a worker imagos. For the sake of flexibility, a stationary imagos may also dispatch user designed messengers. In this case, the system will load the user designed messenger code from the local host, create a thread and add the messenger thread into the ready queue for execution.

In this section, we will discuss the design pattern of system builtin messengers. Each system builtin messenger has a given code name. The following example shows an asynchronous messenger named as `$oneway_messenger`. It is worth to note that this name is the code name, rather than the imagos’s name, because messenger imagos are anonymous.

```prolog
:- begin_messenger
$oneway_messenger(Receiver, Msg) :- deliver(Receiver, Msg).
deliver(Receiver, Msg) :- attach(Receiver, Msg, Result),
    check(Receiver, Msg, Result).
check(\_ received) :- !, dispose.
check(Receiver, Msg, moved(Server)) :- !, move(Server),
deliver(Receiver, Msg).
check(Receiver, Msg, cloned(Clone)) :- !, clone(\_ R),
    R == clone \rightarrow
deliver(Clone, Msg);
deliver(Receiver, Msg).
check(\_ deceased) :- !, dispose.
:- end_messenger
```

When the `$oneway_messenger` is started, it tries to attach itself to the given receiver. Only two possible cases make the `attach` succeed immediately: either the receiver has moved or the receiver has deceased (here we consider the receiver dead if it could not be found through the IMAGO name resolution). For the former case, this messenger will follow the receiver by calling `move` and then try to deliver its message at the new host; for the later case, the messenger simply disposes itself. Otherwise, the receiver must be alive at the current host, thus the messenger attaches to this receiver and makes the receiver ready if the receiver was blocked by a `wait_accept`. 
After having attached to its receiver, the messenger is suspended. There is no guarantee that the receiver will release this attached messenger by calling an accept-type predicate, because the receiver is free to do anything, such as move, back or clone before issuing an accept, or even dispose without accepting messengers. For this reason, a resumed messenger must be able to cope with different cases and try to re-deliver the message if the message has not been received yet and the receiver is still alive.

An interesting case is when the receiver imago clones itself while it has pending messengers. In order to follow the principle that a cloned image must be an identical copy of its original, all attached messengers must also clone themselves and then attach to the cloned image. From the $oneway_messenger program, we can find that after knowing that the receiver has been cloned, the resumed messenger invokes clone and then an if-then-else goal is executed: the original messenger re-attaches to the original receiver and the cloned messenger attaches to the cloned image. The word identical copy refers to the “as is” semantics, that is, at the time an image issues a clone predicate, it takes a snapshot (stack, messenger queue, etc.) to create the identical copy. Therefore, a cloned image will have the same messenger queue as its original, but messengers pending in the queue are new threads representing cloned messengers.

The $oneway_messenger is the most basic system built-in messenger image. It is simple and easy to understand. The overhead of its migration from host to host is only slightly higher than the cost of peer to peer message communication, because the amount of its bytecode and execution stack is very small. It implements asynchronous communication between a sending image and a receiving image. It has the ability to automatically track down a moving receiver. Briefly, it has the intelligence to deliver a message to its receiver in a changing, dynamic mobile world.

Other system built-in messengers for send-receive-reply, multicasting and broadcasting can be designed in the similar pattern. Unfortunately, space does not allow for further discussion of these issues.

5 An Example

In this section, I show a possible IMAGO application which simulates a mobile agent sniffing the price changes in an imaginary TSE_server. For the sake of simplicity, this example is presented with assumptions of services and user interfaces. The program starts from the stationary image stock_monitor which creates a worker image with the name sniffer and an argument involving lists of stocks to be monitored for sale or buy, and then it waits for messengers. Upon receiving a message, the application terminates if the message indicates that the market is closed, or it displays the message otherwise.

When the sniffer starts execution, it moves from the home host to the TSE_server. Upon arriving, the sniffer continues execution by calling split/2 which will examine the given argument list to determine whether a clone is nec-
If the argument involves both **Buy** and **Sale** stocks, the **sniffer** clones itself such that the original sniffs **Buy** list whereas the clone sniffs the **Sale** list.

/* Example: Stationary Imago */
:- begin_stationary
stock_monitor :- create(‘./sniffer ima’, sniffer,

[[s(‘NT’, 26.00), s(‘RY’, 43.00)], [s(‘SW’, 53.00)]],

monitor.

monitor :- wait accept(W, Msg),

display(W, Msg),

monitor.

display(_, complete) :- // print “market closed”

terminate.

display(W, Msg) :- // print W and Msg

beep.

:- end_stationary

/* Example: Worker Imago */
:- begin_worker

sniffer( [Buy, Sale]) :- move(‘TSE_server’),

split( Buy, Sale).

split([], []) :- dispatch($oneway_messenger, queen, complete),

dispose.

split([], Sale) :- !, sniff( Sale, sale).

split( Buy, []) :- !, sniff(Buy, buy).

split(Buy, Sale) :- clone(twin, R),

R == clone ->

sniff( Sale, sale);

sniff(Buy, buy).

sniff(L, Act) :- query(L, Act),

sleep(2000),

sniff(L, Act).

query([], _) :- !.

query([[s(X, Y)|L], Act)] :- database(‘FIND PRICE’, X, Y1), // assumed service

check(X, Y, Y1, Act),

query(L, Act).

check(_, Y1, _) :- var(Y1); // if unbound, market closed

dispatch($oneway_messenger, queen, complete),

dispose.

check(X, Y, Y1, buy) :- Y > Y1, !,

dispatch($oneway_messenger, queen, knock(buy, X, Y1)).

check(X, Y, Y1, sale) :- Y < Y1, !,

dispatch($oneway_messenger, queen, knock(sale, X, Y1)).

check(_).
Now, the \textit{sniffer} will make queries to the stock database periodically until the stock market is closed (a variable is returned to a query). For each stock listed in its argument, the \textit{sniffer} checks if the new price is less than the user's limit. If so, an \textit{oneway_messenger} is dispatched to knock the stationary imago up, otherwise, the next stock will be investigated. The clone, if there is one, will do the same work as described above, except it checks for the condition on sale. Clearly, it is possible that no knock-up messengers would be dispatched if the stock prices could not meet the conditions for sale or buy.

6 Conclusion

The major feature of the IMAGO API is its novel communication model - to deploy messengers to automatically track down agents and deliver messages in a dynamic, changing world. Research on this subject involves two ongoing projects: a detailed specification of the IMAGO API and the implementation of MLVM. Although this study concentrates on the design of the IMAGO system, results will be also useful in related disciplines of network/mobile computing and functional/logic programming community.

Finally, I would like to express my appreciation to the Natural Science and Engineering Council of Canada for supporting this research.

References

Mobile Agents and Legacy Systems: How to Integrate Alternative Communication Paradigms

Antonio Puliafito¹, Orazio Tomarchio², Lorenzo Vita²

¹ Dipartimento di Matematica, Università di Messina
C.da Papardo - Salita Sperone, 98166 Messina - Italy
E-mail: apulia@ingegneria.unime.it

² Dipartimento di Ingegneria Informatica e delle Telecomunicazioni, Università di Catania
Viale A. Doria 6, 95125 Catania - Italy
E-mail: {tomarchio,lvita}@iit.unict.it

Abstract. Over the last few years a large number of mobile agent systems have been developed, both in the academic field and in the industrial one. However, agent technology has hardly been adopted in developing commercial applications, notwithstanding its interesting potentialities. In our thinking, this has been mainly due to the lack of interoperability of agent technology with traditional and common techniques for developing distributed software, and to the fact that it has been often presented as the paradigm suitable for all distributed applications. Starting from this consideration, in this paper we present a model for integrating mobile agent technology into a common distributed object architecture such as CORBA. The implementation of the architecture has been carried out using our agent platform MAP, and the main strength of the adopted approach will be shown.

Keywords: Distributed object-oriented technology, Mobile agents, CORBA.

1 Introduction

The mobile agent programming paradigm has been successful both among researchers and companies [6, 12]. In fact, several agent platforms have been developed during the last few years. Furthermore, more and more complex agent-based applications are available in the areas of information retrieval, e-commerce, and mobile computing. As it is described in [5], the reasons for using the mobile agent technology concern both the benefits that can be obtained for better performances and reliability of systems, and also the benefits arising from an organization of software that could comply with the more sophisticated mechanisms of communication, coordination and synchronization available. Some of the benefits arising from the use of the mobile agent programming paradigm are the reduction of network load thanks to a local interaction with distributed resources, a better fault-tolerance, and the support for operating even in conditions of temporary disconnection from the system [7]. More in general - these are the benefits related with a different organization of the code - we would like to point out how the mobile agent paradigm helps a programmer to model (and therefore to develop)
distributed applications, thus overcoming the traditional client/server communication paradigm [1]. Many efforts have been taken by researchers and companies for defining some standards related with the agent programming paradigms such as MASIF [11] and FIPA [3] and in the creation of platforms implementing them. Notwithstanding this, such technology is not yet perfectly integrated in the process of software production, where the role of technology like CORBA is very important. As we will clarify below, CORBA [18] is an industrial standard that certainly provides several benefits for developing a distributed networked application, but is based on the RPC mechanism. If we try to explain the reasons for this situation, we can notice how mobile agents have always been presented - since the time of their first appearance - as an alternative mechanism to the traditional techniques of distributed programming. This has made the gradual introduction of mobility into an application virtually impossible, not favoring the use of such development model for distributed applications, if we consider that, at the same time, nearly any application implemented by means of mobile agents could be developed according to the traditional client/server paradigm [2, 9].

Thus, starting from the experience made during the last few years with mobile agent systems [17, 16, 10, 8], and being sure of the need for using a model whose client/server communication mechanisms coexist with the code mobility, in this paper we propose an extension of this agent system, which enables us to ”integrate” the client/server and the mobile agent paradigms, in order to:

- exploit the benefits of agent mobility in a CORBA environment, with no need to structure the whole application as an agent;
- access CORBA services from an agent, and thus interacting with existing legacy systems.

The solution proposed has been implemented within the MAP agent platform [17, 8] that (transparently for the programmer) provides with the opportunity of considering an agent as a CORBA service and accessing CORBA services even from within a mobile agent. After reviewing the main characteristics of the various distributed programming models in Section 2, in Section 3 we describe the architecture of the system proposed. In Section 4 we describe some implementation details while Section 5 presents an example where we applied the developed mechanism. Finally, we end our work in Section 6.

2 Traditional distributed programming and mobile code paradigms: a comparison

Distributed applications are traditionally based on the client-server paradigm, where the interaction among processes takes place by means of both message-passing and remote procedure call (RPC). These communication models are synchronous. This means that the client process, after sending the request to the server, suspends, waiting for a reply. This is the paradigm which more recent models of distributed object programming - such as CORBA [18] - are based on. The object model on which the CORBA architecture is based, allows applications to be built in a standard manner using ”objects” as basic building blocks. Therefore, a CORBA based system is a collection of objects that isolates the requestor of services (client) from the provider of services (server) by
a well-defined encapsulating interface. Beyond the object model it uses, CORBA enhances the basic RPC paradigm since CORBA objects can run on any platform, they can be located everywhere on the network and can be written in any programming language that has IDL mapping.

Notwithstanding the benefits and the features provided by this model, the interaction through the network takes place between two objects whose code is statically resident in specific hosts. The concept of ”code mobility” aims to removing this restriction: in fact, thanks to an appropriate runtime system, a software module can be dynamically run on a host where it has not been installed and statically configured (like in the case of the common remote procedures). Several levels of mobility and, as a result, several paradigms, can be defined. This can be done according to the place where the code is resident, where it is run, and the entity that enables it. A classification of such paradigms can be found in [4], where a distinction is made among code on demand, remote evaluation and agent mobility. The mobile agent paradigm, which is the most generic case of mobility, provides that a software module can ”move” from a node to the other of the network, where it can continue running: while doing so, an agent carries its state, as well as its code. In the most generic of the cases, the state enables the agent to resume its execution from the point where it had been interrupted.

This changed perspective provides with some benefits for some application scenarios, in comparison with a typical system based on the exchange of messages [7]: reduction of the network load, asynchronous and independent execution, dynamic adaptation, ability to work in heterogeneous environments, robustness and fault tolerance. However, the agent technology has not yet been successful in the market, notwithstanding its interesting potential. We think that a reason for this situation might be a wrong assumption. In fact, initially researchers thought that the programming paradigm based on code mobility might replace any existing paradigm, since it was valid and convenient in any situation. This was probably a mistake, due to the lack in a thorough examination of the real benefits provided by agents to the specific application scenario [14]. Furthermore, the existing agent systems cannot be easily integrated with the traditional techniques for the development of distributed applications. This way, an application developer is less likely to appreciate the benefits related to code mobility. In fact, a developer currently either has to choose a traditional distributed object development model (for example, according to the CORBA model), or has to structure the application according to the agent model. In our opinion and also according to some recent literature [9], this is the limit to be passed: the agent paradigm has to be a method for supporting the development of distributed applications; it does not have to be an exclusive method to be used instead of more traditional techniques. The rest of this paper is used for defining and implementing a programming environment where mobile agents can be abstracted in a CORBA environment, and an agent can also access the services made available by CORBA objects.

3 Integrating Corba technology in MAP: system architecture

The purpose of the system proposed is that of enabling the application programmer to select the most convenient programming paradigm within the application, by means of a
middleware layer that uses both the CORBA features and the mobility mechanisms. The system has been implemented using the MAP agent system, developed at University of Catania [8], a platform already compliant with OMG MASIF specifications[11]. Further details about the MAP system can be found in [17, 15, 13]. MASIF interoperability is only a first step, in comparison with what has been shown in this paper: in fact, MASIF enabled us to achieve interoperability among different mobile agent systems. Conversely, in this paper we try to achieve interoperability by means of generic CORBA objects.

In particular, we decided to modify and improve the MAP platform, in order to:

– access the services provided by the MAP platform (and created by agents) to independent CORBA entities that originally were not designed for being hosted by the platform, and that have no mechanism for interacting with software agents.

– equip MAP agents with the tools needed for interacting with CORBA objects.

Our idea is the creation of a two-way bridge between the CORBA world and the one of mobile agents. This way, any CORBA object can interact with software agents, and an agent can access a service provided by a CORBA object. In the following two paragraphs we describe how we could create the interaction modes described.

**Activation of MAP agents from CORBA objects**  Our purpose is that of allowing that a service, which has been specifically developed for the MAP (and is therefore based on the agent paradigm), could be accessed from outside the platform, as a CORBA service. According to the CORBA programming paradigm, a service is represented by an object (*CORBA server*) that exports one or more methods that can be invoked by any CORBA client. An entity therefore needs to be created, that could (on one hand) receive the requests coming from the CORBA world and (on the other hand) process them, by activating the appropriate agents (which, from a logical point of view, are the equivalent of methods for a CORBA object). This entity, which will be called *BrokerCorbaToMap*, will need to be able to interact with the MAP platform (by enabling agents and acquiring data from such products) and with CORBA entities (non-agents), which it will export the above-mentioned services to. We therefore need to implement a mechanism that enables to export the services implemented from a MAP platform to the CORBA world. This mechanism has to assure a high level of transparency, so that the CORBA client
does not notice that the invocation of the service might require the creation, the migration and, in general, the cooperation of the agents dealing with the above-mentioned service within the MAP. Thus, from a logical point of view, each CORBA-like service is associated with a team of agents (that actually perform the service), and with an intermediary object (BrokerCorbaToMap). This is an entity that, acting as an actual CORBA server, can:

- Connect to the ORB and wait for any request coming from CORBA-like clients
- Process the incoming request, and enable the specific agent that will deal with the service requested (if necessary, through the cooperation of other agents)
- Wait for the results processed by the agent(s) in charge of the service, and to communicate them to the CORBA client that requested for them

Thus, the integration of the CORBA services within the MAP platform takes place through two programming paradigms:

*Client-Server Paradigm*: the brokerCorbaToMap acts as a server for the calls coming from the CORBA clients;

*Master-Slave Paradigm*: the brokerCorbaToMap is the master. While performing the service requested, it uses agents, which act as slaves.

From the point of view of the implementation, each service introduced in the MAP will be represented by:

- A server object (BrokerCorbaToMap) that, once is enabled, connects to the ORB and can take the requests addressed to it. Furthermore, it can also enable the appropriate agents that will perform the services.
- A pool of agents, whose task is that of performing the services and return the results to the Broker that enabled them.

Figure 2 provides a graphic representation of what we have just described, in the case of three active brokerCorbaToMaps within a MAP platform:

![Fig. 2. From Corba clients to MAP agents](image)

In particular, we can notice how:

- Service A is associated with broker A and two agents (Agent A1 and Agent A2);
- Service B is associated with broker B and one agent (Agent B1);
- Service C is associated with broker A and three agents (Agent C1, Agent C2, and Agent C3).

Finally, we point out that the broker (once it is enabled) is always resident in the memory and is listening on the ORB with regard to the calls related with the service it represents. Conversely, the activation of an agent by its corresponding broker depends on a request for such service.

**Invocation of a Corba object from a MAP agent** In this section we are going to describe the mechanisms created in the MAP for enabling software agents to interact with the CORBA environment, in order to access the services made available by means of an ORB. An agent, which can thus become a CORBA client, will need to be able:

- To obtain the reference to the object which it wants to invoke a service on
- To prepare the parameters of the invocation
- To invoke the service desired
- To process the results of the request

All the features described have been included in an object named *MapToCorbaClient*. The agent that wishes to interact with a CORBA object will need to be able to instance the object MapToCorbaClient, and to configure it according to its needs. As we have shown in Figure 3, MapToCorbaClient’s task will be the invocation of a service on a generic CORBA server present in the ORB which the agent is connected to.

![Fig. 3. From MAP agents to Corba objects](image)

Once the bridges, which allow reaching the MAP environment from the CORBA world (and vice versa), have been built, we can easily imagine and create several types of crossed interactions. In fact, a MAP agent will be able (if has the potential of a CORBA client) to invoke any service provided by a generic CORBA service located in the ORB it is connected to. In a generic case, the CORBA service invoked might request for the interaction (by means of a BrokerCorbaToMap) with other software agents. The same way, we can understand that the agents enabled by a brokerCorbaToMap can invoke a CORBA service present in the ORB, provided by either MAP platforms or by non-MAP entities.
4 Implementation Details

The main classes that implement the system are: BrokerCorbaToMap, AgentCorbaServer, and MapToCorbaClient; unfortunately, for space reason we are unable to provide a complete and detailed description of the implementation.

The introduction of a new service in the MAP implies a minimum effort, which does not require any other skills than the ones needed for mobile agent programming. No code line has to be written for the management of the complex CORBA world, since the class BrokerCorbaToMap is ready for the use.

The class AgentCorbaServer, which derives from the class Agent of the MAP, is the class which all the agents, doing services for a broker, need to extend. The code of an agent designed for the creation of a CORBA service has to be written as follows:

– Providing the class of the agent with the same name as the service that it will implement.
– Extending the class of the new agent from the AgentCorbaServer class (in order to equip it with the above mentioned capabilities).
– If the service represented by the agent has some input parameters, the builder has to be written so that it can accept the service parameters in input.
– Rewriting the exec method with the code that implements the service (as we would do for a normal agent).
– Making sure that the instructions given before closing the method include the return to the original platform (that is, the home), and the return of the results to the broker that enabled it (by invoking an appropriate method of the Broker).

Conversely, the class MapToCorbaClient puts some tools at an agent’s disposal. Such tools allow the agent to interact with the CORBA world. An agent only needs to instance an object MapToCorbaClient, by sending the name of the service and the one of the interface implemented. The MapToCorbaClient will use such parameters for obtaining some information from the Interface Repository; such information will be used while building the request. The main method of the class MapToCORBAclient, as well as the only public method visible from outside, is invoke-method: this method allows to invoke the CORBA service desired within the agent.

5 A case study

For demonstration purposes, we have created a prototype of a distributed holiday resorts reservation system. The basic assumptions are the existence of several holiday resorts providing several services to their customers and of a single point for accessing the system and allowing the user to search and reserve the selected resort. However, each holiday resort has its own software system (which has been developed according to a CORBA paradigm), and the structure of such local systems should not be changed. The agent technology used in this example allows us to deal with the heterogeneity of these systems, interacting with them by means of the above-mentioned mechanisms.

The solution considered, which has been implemented, provides for a decentralization of services towards the local nodes (resorts), even if the presence of a central node
with limited features is maintained. In fact, each local node maintains the database corresponding to the resort it represents, together with all the services concerning the data of a single resort. Furthermore, it allows the management of local reservations. This is the generic CORBA interface implemented by each resort:

```java
interface LocalReservation{
    string localSearch( ...... );
    string localBook( ...... );
    boolean localCancel( ...... );
}
```

Conversely, the central node of the system, where we assume that the MAP agent system is present, and which acts as an access point for the generic user, implements the following CORBA interface:

```java
interface GlobalReservation{
    string globalSearch( ...... );
    string globalBook( ...... );
    boolean globalCancel( ...... );
}
```

In order to invoke the services provided by this node (and therefore to access the system), the user can use a generic CORBA client, or an appropriate Web-based application, or the agent system equipped with the above-mentioned features. In any case, what we will describe below concerns anything that takes place in the system when one of the methods, provided by the GlobalReservation interface on the access node, is called. Figure 4 shows the steps and the agents enabled in the case of the search operation:

- the method `globalSearch` is invoked, according to the user’s preferences (1);
- this method causes the creation of a `SearchAgent` agent (2), which migrates to the sites of the resorts (3,6), and searches for one that could satisfy the user;
- the SearchAgent (thanks to the features introduced in the MAP) will have the opportunity of interacting (in each site) with the local CORBA objects that implement the service of `LocalReservation`, by invoking the localSearch method by means of the MapToCorba mechanism (4-5,7-8);
- then the agent migrates to each site (6), and discards the offers that are not consistent with the user’s requests;
- finally, the agent returns to the home site (9), and returns the results to the globalSearch method that was initially enabled by the user (10).

After obtaining the search results, the user can continue his/her reservation operation. In this case, these steps are done:

- the method `globalBook` is invoked, according to the user’s preferences;
- this method causes a `BookAgent` agent to be created;
- the BookAgent migrates to the selected resort, where it invokes the method `localBook`, thanks to the MapToCorba interaction;
- finally, the BookAgent sends a message to the home node, in order to notify the user that the reservation has been completed.
From this simple application, we can come to the following considerations: 1) the application that we have considered benefits from the agent system in order to search for useful information for the user; this search can be made on heterogeneous sites, which are probably under the control of several organizations. 2) Instead of performing N client/server transactions with the different resorts, the user’s request is included in the agent that can then migrate to the different local nodes independently. 3) the single local nodes of the resorts do not need to change their software systems completely, for redesigning them according to the agent scheme: we can reasonably assume that they export the CORBA methods, since CORBA is a standard for distributed systems. 4) The client application that allows the user to access the system is not restricted: the user might use a CORBA-based application and invoke the methods of the GlobalReservation interface directly with it, or a Web-based application that acts as a mechanism for accessing the GlobalReservation services. Otherwise, an agent application, by means of the MapToCorba interaction, can invoke such services.

6 Conclusions and future work

In this paper we have presented an architecture that enables an agent system to interact with distributed objects developed according to the CORBA specifications. Thus, this system allows to integrate the agent paradigm in the normal development cycle of a distributed software application, by using the mechanism of mobility only for the aspects that can actually benefit from it. The system has been implemented on top of the MAP agent platform, in order to check for the feasibility and validity of this approach. Although the described case study adequately shows the basic features of the infrastructure, future work on these aspects regards the experimentation of the system with wider-scale applications.
Acknowledgement Work carried out under the financial support of Italian Ministry for University and Science and Technology Research (MURST) within the project framework of "MUSIQUE: Infrastructure for QoS in Web Multimedia Services with Heterogeneous Access"

References

Agent-Based Distance Vector Routing

Kaizar A. Amin, John T. Mayes and Armin R. Mikler

Department of Computer Science
University of North Texas
Denton, Texas 76203, USA
{amin, mayes, mikler}@cs.unt.edu

Abstract. Mobile Agents are being proposed for an increasing variety of applications. Distance Vector Routing (DVR) is an example of one application that can benefit from an agent-based approach. DVR algorithms, such as RIP, have been shown to cause considerable network resource overhead due to the large number of messages generated at each host/router throughout the route update process. Many of these messages are wasteful since they do not contribute to the route discovery process. However, in an agent-based solution, the number of messages is bounded by the number of agents in the system. In this paper, we present an agent-based solution to DVR. In addition, we will describe agent migration strategies that improve the performance of the route discovery process, namely Random Walk and Structured Walk.

1 Introduction

Routing, the process of selecting a communication path over which data can be sent in a network, is an important aspect of a communication network as it affects many other characteristics of the network performance. Most of the conventional routing algorithms are based on either of the two shortest path routing strategies, namely, Distance Vector Routing or Link State Routing. This paper focuses on Distance Vector Routing (DVR), an iterative, asynchronous and completely distributed routing algorithm [2]. Certain implementations of DVR such as RIP (Routing Information Protocol) are used widely in many networks [4] as they can be easily configured and maintained [9]. However, it has been shown [3] that a large number of update messages exchanged by adjacent nodes in a network constitute considerable resource overhead. This overhead is inflated due to the fact that many of these messages have little or no effect on the route discovery process. Reducing the resource overhead may allow for DVR-class algorithms to be deployed in a wide range of networks (wireless, ad-hoc) which require a simple routing protocol due to limited availability of resources (memory, bandwidth). Motivated by the need to reduce the resource overhead associated with DVR, and following recent developments in ant routing [2], a new implementation of DVR using an agent-based paradigm known as Agent-Based Distance Vector Routing (ADVR) has been developed.

Agents, Software Agents, Intelligent Mobile Agents, and Softbots are terms, which describe the concept of mobile computing or mobile code ([12], [11]). The
mobile agent paradigm has attracted attention from many fields of computer science. The appeal of mobile agents is quite alluring - mobile agents roaming
the Internet could search for information, meet and interact with other agents
that roam the network or remain bound to a particular machine. Agents are
being used or proposed for an increasingly wide variety of applications, ranging
from comparatively small systems to large, open, complex real time systems.
The agent paradigm offers a rich repertoire of features and lends itself to the
formulation of solutions to computational problems in large distributed infra-
structures. In these types of applications, knowledge of node-based parameters is
often essential to make rational decisions. Load balancing and network routing
are typical examples of such applications. To efficiently route packets through a
large communication network, the constituent network nodes may require topo-
logy information for generating the routing maps or routing tables [7].

In Section 2, a brief overview of DVR is given along with an overview of
different agent movement strategies. The various tools used to simulate the net-
work environment are presented in Section 3. Section 4 gives a detailed analysis
of experiments and results. Section 5 provides a summary of the paper along
with the scope of future work with respect to the utility of agents in distributed
networks.

2 Agents in DVR

Distance vector routing (DVR) algorithms exchange a metric that represents
the distance from a node $n_i$ to any destination $n_j$. Distance is a generalized
concept [5], which may include (but is not limited to) transmission delay on a
link, monetary cost of traversing a link, resource reservation in sending messages,
security level of links/nodes, or reliability measures. In most implementations of
DVR this information (metric) is exchanged among adjacent nodes in the form
of triggered updates, which is initiated when there is a change in the routing
table of one of the neighboring nodes. After receiving the update information
from a neighboring node, a node $n_i$ updates its own routing table in the following
manner:

$$D(i, j) = \begin{cases} 0 & \forall \ i = j \\ \min[d(i, k) + D(k, j)] & \forall \ n_k \text{ adjacent to } n_i \end{cases}$$

(1)

where $D(i, j)$ represents the metric of the best route from node $n_i$ to node $n_j$
currently known to $n_i$. $d(i, k)$ represents the cost of traversing the link from node
$n_i$ to node $n_k$. Any node $n_i$ that receives $D(k, j)$ from a neighbor $n_k$, computes
$D(i, j)$ and integrates this value in its routing table. When the routing table
of $n_i$ is updated, it propagates this change to all its neighbors, which in turn
perform the same algorithm. Therefore, an update in one routing table can cause
a sequence of update messages in nodes throughout the entire network.

While the message activity in conventional DVR can escalate to consume
significant amounts of network resources, the number of messages in ADVR is
bounded by the number of constituent agents in the network. In ADVR, the
exchange of the metrics and the process of route discovery moves from the nodes to the agents [7]. Hence in this approach, the route discovery is manifested in the movement of agents carrying routing information from one node to another rather than the propagation of individual update messages. An agent can be formally described as:

\[ A(i, x, y, R_x, \gamma) \]

where \( A \) is an Agent with ID \( i \) migrating from node \( n_x \) to node \( n_y \), carrying the routing table \( R_x \) of \( n_x \) and using the migration strategy \( \gamma \) to move among adjacent nodes.

In ADVR, agents start at arbitrary nodes and migrate to adjacent nodes using \( \gamma \) as shown by Figure 1. On arriving at a node \( n_y \), an agent \( A(i, x, y, R_x, \gamma) \) updates the routing table \( R_y \) based on the following equation:

\[ D(y, j) = \min(D(y, j), [d(y, x) + D(x, j)]) \quad \forall \ n_j \text{ carried in the agent} \quad (2) \]

where \( D(x, j) \) is an entry in \( R_x \). While equation(2) is based on equation(1), it is performed less frequently in ADVR as compared to DVR. The agent then selects \( R_i \) and migrates to an adjacent node using migration strategy \( \gamma \).

### 2.1 Agent Migration Strategies (\( \gamma \))

It has been shown in the previous section that, in ADVR, agents migrate among nodes, thereby establishing routes for every pair of nodes in the network in a distributed way. Hence, the efficiency of ADVR, in terms of the route discovery, is characterized by the migration strategy of the agents. It is important that the agents migrate intelligently, since an imprudent strategy can severely affect the performance of ADVR. To demonstrate this fact consider the following example of a three node ring graph as shown in Figure 2, with the following migration strategy: While migrating from a node \( n_i \), the agent selects any node from a pool of nodes adjacent to the \( n_i \) at random. However, the agent will refrain from reversing its direction. This strategy assumes that a node would not benefit from consecutive visitations. Intuitively, this strategy would avoid looping between two immediately adjacent nodes. However, this may introduce an indirect looping problem, since, the agent will be forced into a loop (step T2 → step T3 → step T1 ... ) not allowing ADVR to converge (see Figure 2). In general, deploying this scheme for any network topology may cause unnecessary looping and thus degrade the performance. Therefore, migration strategies for
ADVR should be chosen carefully, as it might have severe side effects. An agent migration strategy (\(\gamma\)) can be formally described as \(\gamma_{rw}(x,y,f(\cdot))\) where \(\gamma\) is the strategy to migrate from a node \(n_x\) to node \(n_y\) using the function \(f(\cdot)\) to select \(n_y\). Different agent migration strategies can be formulated by changing \(f(\cdot)\).

This paper proposes two migration strategies, namely, Random Walk (\(\gamma_{rw}\)) and Structured Walk (\(\gamma_{sw}\)).

A Random Walk (\(\gamma_{rw}\)) is an agent migration strategy in which \(f(\cdot)\) is a random function selecting an adjacent node \(n_y\) from a pool of nodes immediately adjacent to \(n_x\). A Random Walk is a useful migration strategy due to its simplicity. It has been shown that, due to its probabilistic nature, a Random Walk will visit all nodes and edges (given infinite time) in a network thereby causing the system to converge [8].

A Structured Walk is a movement strategy which exhibits a deterministic behavior based on some criteria, such as congestion levels, topological information, and past visitations. In a Structured Walk (\(\gamma_{sw}\)), \(f(\cdot)\) is a function that selects a node \(n_y\) for migration such that \(n_y\) satisfies the condition of minimizing or maximizing some decision criteria. For example, a Structured Walk may use \(\min(v)\) as a decision criterion, where \(v\) represents the frequency of node visitations by an agent. Efficiency of the Structured Walk depends on the calculation of \(v\) for every node. In what follows, we describe three different ways for calculating \(v\), based on visitation of nodes, visitation of edges and a combination of both (Least First Walk).

When the selection criterion \((f_{Node(\cdot)})\) for \(v\) is the number of node visitations, we refer to it as a Structured Walk on Nodes. In this case, upon visiting a node \(n_x\), the agent increments the visit count \(v_x\) of that node. At the time of migration of the agent from a node \(n_y\) to its neighbor, the agent selects the adjacent node \(n_z\) which has \(v_z = \min[v_i] \forall n_i\) adjacent to \(n_y\). When there is more than one node with the same \(\min(v)\), the agent selects one at random. This scheme relies on the assumption that a node with fewer visitations will discover more routes when visited.

When the selection criteria \((f_{Edge(\cdot)})\) for \(v\) is edge visitations, we refer to it as a Structured Walk on Edges. Whenever an agent traverses an undirected edge \(xy\), connecting nodes \(n_x\) and \(n_y\), it increments the visitation count \(v_{xy}\). At the time of migrating from a node \(n_y\) to its neighbor, the agent selects an adjacent node \(n_z\) for which the connecting edge has a minimum \(v\), i.e. \(v_{yz} = \min[v_{yi}] \forall \)
\( n_i \) adjacent to \( n_y \). As with \( f_{\text{Node}}(\cdot) \), multiple \( \min(v) \) are resolved at random. Intuitively, a Structured Walk on nodes might improve route discovery, since in every step, the agent moves to a node that is either unvisited or least visited. This however may not be true, since route discovery involves finding the shortest path between nodes. Hence, it is important to explore all the paths that exist, making it beneficial to traverse all the edges (Structured Walk on Edges) in the network.

A combination of the above mentioned methods is referred to as a Structured Least First Walk (\( f_{\text{LFW}}(\cdot) \)). This strategy is a slight modification to the Structured Walk on Edges. Whenever an agent traverses a node or an edge, it increments the respective visitation counts. At the time of migration from a node \( n_y \) to its neighbor, the agent selects an adjacent node \( n_z \) for which the sum of the visitation counts \( u_z \) of that node and the visitation count of the connecting edge \( v_{yz} \) is minimum. This is formally expressed as:

\[
v_{LFW_{yz}} = u_z + v_{yz} \\
v_{LFW_{yz}} = \min[v_{LFW_{yz}}] \ \forall \ n_i \text{ adjacent to } n_y
\]

Structured Least First Walk will aid multiple agents to coordinate their actions when traversing the network. Structured Least First Walk has been used for the experiments conducted as a part of the analysis of the ADVR.

## 3 Experimental Design

In this section, we describe our simulation environment and present the results of our experiments. The experiments focused on providing a comparative analysis of ADVR vs. DVR. The simulation results indicate that agents with the most rudimentary of intelligence will bring the network to a connected/converged state. In addition, it is evident that although single agent systems will bring the network to a connected/converged state, multi-agent systems will take advantage of intrinsic parallelism and improve the connection/convergence pattern. The design and deployment of smarter agents improves the connectivity/convergence pattern, however, care must be taken when choosing an agent migration strategy.

Our performance analysis was based on the following criteria:

- **Connectivity**: The state of a network when every node in the network has discovered a path/route to every other node.
- **Convergence**: The state of the network when every node knows the optimal path (minimum cost) to every other node.
- **Message Efficiency**: The proportion of messages that cause an update of routing tables.

### 3.1 Tools

To investigate the properties of agents in DVR, an event driven simulator and graph generator have been constructed. The simulator is based on an object-oriented paradigm [13] and includes methods methods for DVR, single agent
ADVR and multi-agent ADVR. The simulation model, as depicted by Figure 3, contains the following objects:

- Simulator - simulation engine for scheduling and dispatching events.
- Graph - container object that supplies a global view of all vertices and edges.
- Vertex - representation of a node/router that provides a routing table and methods for DVR.
- Edge - representation of a physical link between two vertices with an associated link cost.
- Agent - representation of a single agent containing methods for ADVR.
- Events - (Graph, Vertex, Edge and Agent Events) wrapper objects facilitating communication between the respective objects and the simulator.

![Fig. 3. Simulation Model](image)

A network is represented as a graph \( G(V, E) \) that is generated by the graph generator. The graph generator constructs pseudo-random, connected, undirected graphs with \( V \) nodes and \( E \) edges, given a random seed as input. A graph \( G(V, E) \) is generated in a two step process. First, the graph generator builds a random spanning tree containing \(|V| - 1\) edges as shown by Figure 4 lines 7 - 11, hence ensuring that the graph is connected. Secondly, it adds \( \epsilon - (|V| - 1) \) random edges from \( S - E[G] \), where \( S = \{u \times v | u \neq v; u, v \in V\} \), to make \( \epsilon \) edges in total. Features to control the average node degree of \( G \), \( \delta(G) \), have been implemented, however, the details of the features of the graph generator are beyond the scope of this paper.

### 3.2 Experiments

In the experiments conducted, we have made certain underlying assumptions. We assume the network to be stable, i.e. edges and nodes are neither added nor deleted. The analysis does not cover the performance of the network after convergence of routing tables. The results of experiments that address link or node failure are beyond the scope of this paper and are discussed elsewhere.
MAKE-GRAPH(V,e)
1. \( G \leftarrow V \)
2. \( P \leftarrow \{\} \)
3. \( D \leftarrow V \)
4. \( v \leftarrow \) randomly chosen vertex of \( D \)
5. \( C \leftarrow \{v\} \)
6. \( D \leftarrow D - \{v\} \)
7. for each random \( v \in D \) do
   8. \( u \leftarrow \) random vertex in \( C \)
   9. \( E[G] \leftarrow E[G] + \{\{u, v\}, \{v, u\}\} \)
10. \( D \leftarrow D - \{v\} \)
11. \( C \leftarrow C + \{v\} \)
12. if \( u \neq v \) then
    13. \( P \leftarrow P + \{\{u, v\}\} \)
14. while \( |E[G]| < \varepsilon \) do
15. \( \{u, v\} \leftarrow \) random edge in \( P \)
16. \( E[G] \leftarrow E[G] + \{\{u, v\}, \{v, u\}\} \)
17. \( P \leftarrow P - \{\{u, v\}, \{v, u\}\} \)

Fig. 4. Pseudo-random Graph Generation Algorithm

For the simulation of DVR, we only consider triggered updates. However, timed updates will increase the resource overhead and further reduce the performance of DVR. Agent population is assumed to be static. Our experiments are based on three types of networks, namely small, medium and large. Small networks have 25 nodes, medium sized networks have 60 nodes and large networks have 100 nodes. Density of the network is defined by the number of links. A dense network \( (G(V, E)) \) has number of bidirectional links \( |E| \) closer to \( \frac{V^2 - V}{2} \) whereas, a sparse graph has links closer to \( |V| \). Simulations were parameterized on the basis of network size, network density and simulation type (DVR or ADVR).

From equation (2), we see that an agent updates a routing table only if it has a lower cost to the destination. Therefore, on every update ADVR will bring the routing table closer to convergence. ADVR is characterized by reduced concurrency, as compared to DVR. The degree of concurrency in ADVR is bounded by the number of constituent agents. Figure 5a compares the convergence pattern of DVR vs. ADVR with different number of agents. DVR has a better initial convergence than ADVR, which is explained by the fact that DVR broadcasts messages to all its neighbors. On the other hand, ADVR is marked by the migration of agents which restrict the parallelism to the number of agents in the network. Hence the initial convergence rate for ADVR is proportional to the number of agents. Further, we observe that although the agents have a slow initial convergence, they compensate for it with their intelligent migration strategy. An important aspect for ADVR convergence is the agent population. Since the number of agents dictate the degree of parallelism of the algorithm, a large number of agents would exhibit better performance. However, the re-
source overhead increases proportionally with the size of the agent population. Therefore, performance and resource overhead constitute a tradeoff that must be carefully balanced by selecting an appropriate agent population. A characteristic of ADVR performance is the long convergence tail. This tail is due to the fact that there may be a small number of nodes in the network that have not yet converged. Their routing tables reflect a cost that deviates from optimal by δ. The agents migrate among nodes until the network has converged. The size of the convergence tail is inversely proportional to the number of agents in the network. While this appears to be a drawback, in a realistic network environment, the total routing cost exhibit fluctuations larger than δ.

Route discovery plays an important role in network performance with respect to fault tolerance. Hence, it is crucial to evaluate any routing algorithm with respect to the speed at which routes between any two nodes can be obtained. Figure 5b depicts ADVR’s progress in identifying routes in the network. The route discovery process for ADVR improves with an increase in the number of agents by exploiting concurrency.

As mentioned earlier, this paper aims at reducing the message overhead incurred by DVR. The large number of messages generated by DVR can be attributed to the highly concurrent and completely asynchronous behavior of DVR. In ADVR, the number of messages in the network is bounded by the number of constituent agents. Figure 5c compares the message efficiency for the two approaches. It indicates that the proportion of effective messages in ADVR is significantly higher as compared to DVR. Therefore, ADVR is suitable for wireless networks, with low resource (bandwidth) availability [10]. Intuitively, reducing the concurrency in an algorithm, reduces its performance. However, an appropriate migration strategy will improve the message efficiency, hence, ADVR can achieve superior performance with only $c$ agents ($c \leq n$, where $n$ is the number of nodes in the network).

As shown in Figure 2 that agent migration strategies can cause considerable side effects thereby delaying convergence of ADVR. Both, Random Walk and Structured Walk, can be applied to different classes of applications. While it can be shown that the two schemes yield comparable convergence, Structured Walk outperforms Random Walk migration with respect to the rate of route discovery (see Figure 5d). Therefore, a Structured Walk can be used in networks where early route discovery is crucial whereas, a Random Walk is applicable in systems which require a simple implementation.

4 Summary and Future Work

In this paper, we have described an agent-based paradigm for a Distance Vector Routing scheme (ADVR). In ADVR, intelligent mobile agents are the principle carriers of update messages transmitted between routers for the purpose of route computation. One of the major disadvantages of conventional implementations of distance vector routing algorithms is that their corresponding resource overhead is generally unbounded. That is, the overhead due to update messages will
increase proportionally with the size of the network. In the proposed ADVR, the messages are replaced by a population of agents. Hence, the overhead is bounded by the number of agents. However, by limiting the number of agents in order to control resource overhead, the degree of concurrency which the algorithm can employ is restricted as well. We have conducted a number of experiments to analyze the performance of an agent-based distance vector routing scheme. In particular, we have focused on agent migration strategies, agent population, convergence behavior, route discovery and message efficiency.

This paper has introduced the concept of a Structure Walk during which agents utilize specific runtime information which allows the agent(s) to migrate through large parts of the network efficiently. We have provided an example to demonstrate the significance of choosing an appropriate migration strategy to guarantee route table convergence. Through a number of carefully designed experiments, we have shown the quantitative improvements in route discovery.
and cost convergence by increasing the number of agents in the constituent agent population. The convergence behavior, as well as the rate at which new routes can be discovered, have been compared to a conventional implementation of distance vector routing. Last but not least, we have quantified and compared the message efficiency of ADVR and DVR.

Ongoing research is focusing on fault tolerant routing in dynamic networks, tackling the *Counting to Infinity Problem*, and exploitation of dynamic agent population. While these issues are certainly important, their discussion was beyond the scope of this paper and will appear in a future publication.

References

5. RFC 1058
High Availability Connection Management via Software Bus and Mobile Agent in Network Management Environment

LI Yewen and MENG Luoming

National Laboratory of Switching Technology and Telecommunications Network, BUITP Mail Box 198, Beijing University of Posts & Telecommunications (BUITP) 10 Xi Tu Cheng Road, Hai Dian District, Beijing, 100876, China B9973041@bupt.edu.cn lmmeng@bupt.edu.cn

Abstract. This paper analyzes the requirements of reliability connection in network management environment (NME). It points out the shortcomings of existed fault tolerant system and high availability (HA) system in NME. In order to address the connection reliability problem, new concept of HA connection is proposed. In order to assure the HA connection in NME, this paper proposes a provision of software bus theory which is composed of basic concept, link model, implementation model and network implementation model. Then, an implementation of software bus based on message mechanism and mobile agent supporting HA connection is discussed in detail. At last, performance of this system is discussed and evaluated.

1 Introduction

With the rapid development of network, network management systems (NMS) are playing more and more important role. An effective NMS can assure a network to run normally, economically, and reliably. However, when we construct NMS, the reliability of the NMS itself is also very important. How to ensure the reliability of the NMS has been paid great attention by network management developers.

The NMS is typical of a system that uses distributed technologies. One feature of NMS is that there is a large amount of data communication between the management system and the managed system. Logical units of the management system also need to communicate with each other for exchanging management information. Tests have shown that nearly half of the running time of the NMS is used to transfer data. So, to guarantee the High Availability \cite{1,2} of that communication is the main task to guarantee the reliability of the NMS.

To address these needs, Fault Tolerant (FT) \cite{3} systems comprised of specially designed, redundant hardware, tailored operating systems, and highly customized
application software have been developed. While effective for their specialized uses, they are usually expensive, difficult to maintain, and require scheduled downtime for operating system software and other maintenance upgrades. In addition, FT systems only provide protection from localized hardware-related failures. However, nearly half of the system outages are the result of non-hardware-related problems, especially in distributed systems. And these are just the unplanned outages. As a result, what most enterprises need is not only a more affordable safety net, but one that provides a broader level of protection than hardware redundancy alone can achieve. This protection has come to be known as HA.

The common HA solution is a HA software clustering solution that protects critical information services through monitoring, restarting, failing over, and recovering all critical components in clusters of two or more servers. This flexible software was designed to handle network and data integrity problems found in networked client/server environments. The software keeps client/server operations up and running by providing automatic failure detection, eliminating all single points of failure and assuring availability to servers and networks. The most common use of HA is to improve HA of NFS, RDBMS, Web, and communications server etc. Based on the above analysis, we know that the existing HA system focuses on the protection of the server to improve the reliability of a system. That is to say the existing HA system can only guarantee the reliability of the applications processing. It cannot ensure the communication high availability of all the applications in a system. In NMS, this will lead to data loss and network management quality reduction. From these, we know that the exiting HA system cannot satisfy the needs of the NMS that needs HA connection. To address this problem, we propose new concept of HA connection in NME and design a kind of software bus based on message mechanism and mobile agent.

The remainder of this paper is organized as follows. Section2 gives the definition of HA connection. Section3 presents the provision of our software bus theory and is followed by HA software bus design and implementation in section4. Performance evaluation is presented in section5, and conclusions and future work in section6.

2 The Concept of HA Connection

The concept of HA connection is a new concept we proposed which is applied in NMS environment. HA connection means the communications between network management applications are HA. In complex and heterogeneous network management environment, network management applications may work on different hardware and software platforms. In such an environment, in order to communicate, network management applications have to deal with many complex things such as operating systems, word length, process id, network address, network protocol, network parameters and the like. But all of these should be hidden from the applications. One application only needs to know the name of other application to communicate with it. All of the communication details are transparent to them. The system can deal with interruptions caused by various situations, so the applications perceive the connections between them to be functioning normally. This is very important to the complicated and multivariate NMS. All of these requirements can be concluded as follows.
1. Connection Transparency. The application itself does not deal with the details of the communications.
2. Location Transparency. The communications have nothing to do with the physical situations of the applications.
3. Calling Method Transparency. No matter if it is a local call or a remote call, a uniform format is used and has nothing to do with the environment.
4. Relocation Transparency. Changing the interface of an application doesn’t influence other interfaces with which it is communicating.
5. Failure Transparency. This masks the error and the failure-over procedure from the applications in order to improve the system availability.
6. Migration Transparency. In order to gain load balance, we need reconfiguration sources. In certain conditions processes may migrate actively. All of these should be transparent to the peer with which it is communicating.
7. Persistent Connection. The connection between the applications may be interrupted in many situations. For one, in order to reach the load balance, the processes may migrate. After the migration, they need to reestablish lost connections. The other related communication applications should not know these procedures. For another, a process may crash. Later the process or its backup may restart under certain conditions. The restarting process should reestablish all lost connections. All these procedures should be transparent to the related applications which perceive all the connections as working normally until it is warned that the connection has been lost.

3 The Framework of Software Bus

There are many papers about software bus [4–7] such as Object Request Broker (ORB). However, in CORBA system based on ORB, the reliability of connection between applications is only depending on the reliability of the transport layer can provide [8]. In practical use, extra heavy work is need to assure the reliable communication [9]. In order to address the HA connection in NME, this paper introduces a provision of software bus aimed to address the HA connection problem and gives its implementation detail.

![Fig.1. Connection between Software Units](image)
### 3.1 The Concept of Software Bus

The key of designing a software system is to design the system’s structure. The key of a system’s structure is the unit of this system and the relationship between them. Because the relationship between those units is very complicated, so the technologies, which deal with that relationship, are also very complicated. Traditional method to deal with the relationship between units is to define the interface between them. Because the relationships between those units are very complicated, so the interfaces between them are also very complicated. Fig. 1 illustrates the complexity of the connections between software units.

The basic idea of software bus is not to define the interface between units directly, but to define a kind of “junction piece” to connect them. There is one kind of hardware bus technology in computer system. If we can deal with the relationship between software units like what hardware bus has done, the adaptability and scalability of software system could be improved greatly. The complexity of relevant technology will also be controlled. From the viewpoint of bus, the junction piece can be looked as “software bus”. Based on the concept of software bus, we can define the interface between the software unit and software bus. The advantage of using software interface is that we can change the complicated work, which deals with complicated relationships between units, into a simple work which only deals with the simple relationship between software units and software bus. This method, which deals with the relationship between software units, can also be called software bus. Fig. 2 illustrates its concept.

![Fig.2. Concept of Software Bus](image)

![Fig.3. Link Model of the Software Bus](image)
3.2 The Link Model of Software Bus

Because the software bus can’t provide a channel to exchange data like hardware bus, so related software entities should be used to implement the functions of the software bus. The interface between software units and entities of software bus is called software bus interface. In order to improve the adaptability of software bus, we can adopt the middle service layer of OSI reference model. In order to reduce the implementation complexity, we can simplify the OSI middle service layer. We can use only the “Request” and “Response” as the link model of the software bus as Fig. 3 illustrates. Because the “Request” and “Response” is a kind of relationship between two entities, so the entities of software bus only act as a transferring station to the “request” and “response”.

3.3 The implementation model of software bus

In order to support distributed processing, which means to implement the connection between those entities connected to the software bus, the software bus entities should provide transparent processing function. In other words, the software bus should support the communication only via the application’s name. The method to implement this goal is presented as follows.

- Arrange a sub functional entity of software bus, which is called Element-side (ES), in a software unit end.
- Arrange a sub functional entity of software bus, which is called Bus-side (BS), in software bus functional entities end.

After having done all of the above, we can use application program interface (API) to fulfill our targets. Fig.4 illustrates its basic idea.

![Implementation Model of Software Bus](image)
3.4 The Network Implementation Model of Software Bus

There are many advantages to adopt software bus in software system, which has ES and BS module. First, they provide API for application. Second, they make it very easy for the applications to communicate with each other. Third, the software bus can be implemented in network environment. Especially, the network implementation is transparent to applications. In other words, the upper applications don’t know whether the ES and BS are implemented in network environment. If this network environment could support various hardware and software platforms, it could support connection between those applications, which act on different hardware and software platforms. Fig. 5 illustrates this concept.

The network implementation can be whole network implementation or part of network implementation. In other words, parts of ES and BS are connected via network environment, other parts of ES and BS are connected directly.

![Diagram](image)

**Fig. 5.** Network Implementation Model of Software Bus

4 Implementation of Software Bus

The implemented software bus is called high availability message software bus (HAMSb) which is illustrates in Fig.6.In Fig.6, the Msg Software Bus (MSB), which is in fact one part of the Bus-side of software bus, and applications that directly register on it compose one HA management field. The whole system is composed of several HA management fields. The MSB has message route function between
different MSBs with which communicate each other via redundant physical link. The BS is composed of all MSBs. The MSB can be added into the BS as need dynamically and feasibly. The Element-side of software bus is added between transport layer and application, if we use TCP as transport layer, or above the presentation layer in OSI stack. The interface between the unit of application and the unit ES is called the interface of ES. The MSB plays as both managing station and transferring station. Applications must communicate with each other via the software bus that is composed of BS and ES.

**Fig. 6.** Distributed Software Bus Model

**Fig. 7.** Model of Software Bus
4.1 Message Format Definition

The format of the communication message is defined as follows.

\[
\text{<message> ::= <message head> <parameterList>}
\]

\[
\text{<message head> ::= <msgInvokeId> <msgType> <sourceName> <targetName> <parameterNumber> <encodeType> <escape>}
\]

\[
\text{<parameterList> ::=}
\]

\[
\text{<parameterNameList> <escape> <parameterTypeList> <escape> <parameterValueList> <escape>}
\]

\[
\text{<nameList> ::= <identifier> *}
\]

\[
\text{<parameterNameList> ::= <nameList>}
\]

\[
\text{<parameterTypeList> ::= <nameList>}
\]

\[
\text{<parameterValueList> ::= <nameList>}
\]

\[
\text{<msgInvokeId> ::= <identifier>}
\]

\[
\text{<msgType> ::= <identifier>}
\]

\[
\text{<sourceName> ::= <identifier>}
\]

\[
\text{<targetName> ::= <identifier>}
\]

\[
\text{<parameterNumber> ::= <identifier>}
\]

\[
\text{<encodeType> ::= <identifier>}
\]

\[
\text{<escape> ::= <character>}
\]

Notes:
1. Identifier and character are basic lexical units.
2. Message is composed of fixed part and changeable part. The fixed part is the messageHead. The changeable part is the parameterList.
3. In the fixed part, following contents are described: the message type, the sequence number of the message, the source address and destination address of the message, the encoding method, the number of parameters and the decollator of the message etc.
4. In the changeable part, the parameters of the message are described. Each message is composed of name, type and value. The kind of the message and the parameters can be defined by applications.

4.2 Structure of the HAMSB

Fig.7 illustrates the structure HAMSB which has one MSB. In Fig. 7, the HA-con layer is the ES described in section3.3, which is one part of the software bus. It provides services such as communication by name via API. The software bus entities (SBE) are corresponding to one MSB described above which has many core software entities. The communication is actually implemented by the HA connection layer and SBE which is part of BS.

The SBE is the key of this system, It includes following software entities: the HA-Agent of Bus Side (BSHAA), the Message-Queue (MQ), the Message-Queue-Manager (MQM), Management Module (MM), channel protocol (PP), Mobile Agent Environment and Database etc. BSHAA has the following functions: safety
management, application registering, sending and receiving messages, broadcast service, event-report, call-back mechanism and managing connection etc. The function of managing connection includes watching the availability of the connection and reestablishing the connection lost and establishing connection between the MSB. MQ is a queue which stores the messages sent to it. There are two kinds of messages. One kind is a signaling message, the other is user message. The MSB uses signaling message. The user message is delivery through the MSB transparently. MQM is responsible for managing the messages in MQ. A MQM manages all of the queues in one HA management field. CP is the interface to the specific platform. The mobile agent (MA) is controlled by MM. MA makes the MSB having intelligent \[10\]. When MSB startup, MM will create MA to collect other MSB’s information such as CPU load, free memory and mapping table etc. This is useful for message transferring between multi MSBs and for BS fault tolerance. Because the number of MSB is limited, MA often knows other MSB address. We use IBM’s Agent Building Environment \[11\] and IBM’s Java Aglet API (Aglet Software Development Kit)\[12\] to implement MA. The Agent Transfer Protocol (ATP/0.1) and default port 434 for mobile agent is used. The introduction of MA makes the BS more scalability. When one MSB is overload, we can add another MSB. When one MSB bankrupt, other MSB can take over its function. So this method improves system’s reliability.

4.3 Message Delivery

This system functions as follows. All the applications must register with the bus when it starts. The contents of the registration include the name of the process, the physical address, the interface parameters etc. BSHAA should verify the registration. The BSHAA only notifies the MOM to establish a MQ for the connection that passed the verification. MQM also establishes and maintains a mapping table for the application whose contents include the identifier of the MSB, process name, MQ name and interface parameters etc. The contents of the table can be refreshed through which the system obtains relocation capability. Having registered successfully, the application can send messages to its peers. BSHAA receives the messages and checks the destination of the messages from the message head. Then the messages are sent to the corresponding MQ after having inquired the MQ name in the mapping table. When message arriving at MQ, An acknowledge (ACK) is sent to the source ES and a system interrupt will be raised. Then, a predefined routine is executed to send the messages in the MQ to its destination application. Thus one message has been sent from the sender to the receiver. When destination ES receive the message, it will send an ACK to BS.

In order to guarantee the integrality of the data, a three-step buffer technology is adopted where the message is stored in the sender, the MQ and the receiver. The messages in MQ are also sent to permanent media such as file systems or databases. At the same time, each message is given a permanent ID, through which it can tolerate the abrupt failure of the process or the interruption of the connection or the failure of the BS. When MSB receive ACK from destination ES, it will delete the message stored in permanent storage.

There are many reasons leading to connection breakdown. No matter what causes the connection failure, the MSB will store received messages and continue to receive
the message from the related connection. BSHAA will do its best to reestablish the connection it lost or wait for the registration of the restarted process. When the connection is reestablished, BSHAA will send out messages as before. But the MSB storage is not unlimited. When endpoint failure, the MSB can continue receive message from relative connection. Extra measure is needed to prevent the buffer overflow, which perhaps leads to MSB bankrupt. This cascade will not happen. Because, at this time, those received messages are stored in database. Extra Disk Array is shared by MSB. The MQ will not over flow. When prearranged time passing, and the connection is not restored, an event-report is sent to relative ES and the relative messages will be discarded.

Each MSB will form a mapping table, which contains the information about all the registering applications via the information sent by MA. For messages needing to be delivery across fields, BSHAA will send them to corresponding BSHAA based on the contents of the message head. When one MSB crashes and cannot be restored, another MSB will detects this situation via mobile agent and reestablishes the connections lost. If connection cannot be reestablished, the event-report will be sent to the related applications.

5 Evaluation

From the above introduction, It is clear that the HAMSB is one kind of message-oriented middleware (MOM)[13~14]. Because it adopts message mechanism and message queue. However, it is different from ordinary MOM [15~17]. HAMSB has its own characteristics.

(1) Both BS and ES of software bus are transparent to applications. The application only needs to know the name of another application to communicate with each other.

(2) The communication established is bi-directional and Support broadcast, multicast and event-report.

(3) The software bus adapt to the operation model of manager/agent in NME that needs synchronous and asynchronous communication.

(4) The software bus makes the connection between applications high availability.

(5) The MSB, which is part of BS, can be added dynamically and feasibly as the need to release the load of MSB. MA is used in MSB to find the information about other MSB.

(6) The MSB is fault tolerant. When it is restarted again, it can reestablish all the lost connections on its own initiative and deliver the message stored in persistent storage.

Our MSB is implemented on SUNUltra60, which is composed of 2x300MHz CPU, 256M memory and 4.2G disk. The operating system is Sun Solaris 2.6. The persistent message is stored in Informix Online Dynamic Server Version7.2. However, this method still leads to extra delay. We have tested the delay caused by MSB under normal connection condition, which is shown in table1.
Tests have shown, MBS introduces extra delay which is influenced by the message size each time be sent. When message size is less than 1K, the max delay is 31ms. When message size is 10 K, the max delay is 330ms. When connection error occurred, file I/O will cause more delay. However, this delay is tolerated and worthy of the reliability it has obtained for network management applications.

6 Conclusions

This paper has discussed a method to implement HA connection in NME. The implemented system adopts software bus theory. The basic concept of software bus, the link model, the implementation model and the network implementation model of software bus are presented. The detail of the implementation of the software bus based on message mechanism and mobile agent supporting HA connection is discussed. The implemented system was then evaluated and analyzed, revealing that in order to assure the HA connection and the delivery reliability in NME, this delay is tolerate. This model has been put into practical use in China national mobile network management system \[18-20\] successfully. Further work will be aimed at to improve the system’s performance. We should find a balance point between the HA connection property and system’s performance.

References

Multipoint-to-Point Routing With QoS Guarantees Using Mobile Agents

Sergio GONZALEZ-VALENZUELA*, Victor C. M. LEUNG, Son T. VUONG

1 Department of Electrical And Computer Engineering
The University of British Columbia
2356 Main Mall, Vancouver, B.C. Canada V6T 1Z4
E-mail: {sergiog, vleung}@ece.ubc.ca

2 Department of Computer Science
The University of British Columbia
2366 Main Mall, Vancouver, B.C. Canada V6T 1Z4
E-mail: vuong@cs.ubc.ca

Abstract. To overcome the shortcomings of existing IP networks and to facilitate the overall quality-of-service (QoS) provisioning in the near-future networks, new technologies such as Multi-Protocol Label Switching (MPLS) and Differentiated Services (Diffserv) have been proposed for support of differentiation of classes of services and guarantee of QoS. Diffserv and MPLS, however, require improved capabilities from the current routing algorithms. In this paper, we investigate such an improvement by developing algorithms for determining the optimal multipoint-to-point (mp2p) routes through the use of mobile software agents. We present an mp2p routing scheme using a mobile intelligent agent system, called WAVE. The agents work in a highly distributed and parallel manner, cooperating to determine optimal routes in an mp2p connection scenario. This work aims at closing the gap between the theoretical routing research based on mobile agents, and practical routing requirements for real world networks that are likely to be deployed during the forthcoming years.

1 Introduction

Despite the rapid advances in networking, it is surprising to observe that routing has not grown in parallel when compared to the growth of other networking technologies. A number of extensions and patches have been proposed for current routing schemes in order to keep up with the arising needs in data networks [1], [2].

It would be reasonable to predict that future requirements in networks will impose greater demands on the performance of the routing protocols, as well as a higher computing burden on network nodes, making the complexity of the routing computations intractable [4]. To better understand the problems that current routing schemes will have to cope with, it is imperative to determine possible ways in which new technologies are likely to be configured to obtain the best performance from their mutual interaction. A conceptual model showing some possible supporting elements to achieve QoS at the network layer in the near future is shown in Fig. 1.

* This work supported in part by Conacyt México

© Springer-Verlag Berlin Heidelberg 2001
This document offers an alternative approach to address routing issues by means of mobile agents’ technology. Such approach is presented in the following manner: section 2 presents both the fundamental network-layer concepts of QoS control for the next-generation Internet, and a brief discussion in support of previous proposals on Diffserv over MPLS. In section 3, a short review of the mobile agents’ paradigm is presented, with an emphasis on a novel powerful mobile intelligent system, called WAVE. Section 4 proposes an mp2p routing scheme for use with MPLS and Diffserv by describing a routing algorithm based in WAVE. Finally, Section 5 provides some concluding remarks and offers suggestions for future work.

2 QoS Provision In The Future Internet

Firstly, this investigation focuses on determining a plausible evolution scenario of the Internet backbone. Although recent proposals have been made on the use of novel mobile agents’ technology to support QoS, other technologies and standards are also being developed for deployment for a full QoS-capable environment in the near future. Therefore, instead of proposing an entire QoS architecture, our research takes a proactive approach by considering technologies that already in an advanced state of research and development for support of QoS-control at the network layer, and applies a powerful mobile agents’ system to investigate the issue of routing.

Both Diffserv and MPLS technologies are widely accepted as viable solutions for the next-generation Internet. MPLS [3] emerges as a natural evolution from the label-swapping paradigm, providing data-manageability improvement that goes beyond of those previously offered by technologies such as ATM and Frame Relay. MPLS provides the ease to link together different protocols between layers 2 and 3 in the OSI model. Furthermore, it efficiently supports complex networking tasks, such as: explicit routing, traffic engineering and VPN design [3], [5], [7]. Although the overall operational mechanism of MPLS is a complex one, the basic idea behind this new technology is fairly straightforward. In MPLS, data packets are labelled and switched throughout the network according to a predefined agreement between network nodes [3], [7]. A key enhancement of MPLS relies in its ability to perform an organized management of data flows by strategically assigning labels to each flow. It is this systematic procedure the one that facilitates the implementation of the networking tasks previously mentioned. Specific sets of data streams can be strategically assigned.
with labels in an attempt to group them into Forwarding Equivalence Classes (FECs) [3], according to predetermined forwarding premises (e.g. scheduling precedence, destination, source, etc). Moreover, FECs can be further grouped until a desired level aggregation is achieved. When done appropriately, this scheme provides scalability support for technologies dealing with QoS-control at the network layer. Such is the case of DiffServ [8], which has been most favoured as a viable QoS-mechanism to implement due to its superior scalability features [6], [9], [10]. Proposals have been made where MPLS can be used as a way to organize and transport data flows that share similar scheduling precedence. The aggregated-flow scheme of DiffServ not only reduces the flow state overhead, but also enhances the performance of MPLS by reducing the number of labels to be managed [9], [11]. All the above reasons support the initiative taken to presume that a QoS-control scheme based on DiffServ-over-MPLS has a good chance of being realized, assuming that the nodes in the network are hardware-capable of performing data aggregation by means of current ATM technology through either Virtual Circuit or Virtual Path merge.

It is this data-aggregation notion the one behind the concept of Multipoint-to-Point type-of connections, which are aimed at defining a tree-like path to be used as a manner to establish a shared route among a number of edge nodes in an autonomous system (AS). Thus, the objective of grouping such connections is to achieve an efficient method to better manage incoming data streams that share common characteristics. An example of this scenario is shown in Fig. 2.

An arbitrary number of these trees might be created by proper label assigning under an MPLS environment, according to the QoS characteristics and destination of data. The mp2p scheme greatly resembles its counterpart: the Point-to-Multipoint (p2mp) type-of connection, also known as multicast tree. However, in a multicasting connection all the end peers are involved in a common session. In this case, both the data transferred and the generating entity is identical for all destinations. This may not be the case for the mp2p scenario, where the entities forwarding data to a common root node in the tree are not necessarily participating in a common session of data transfer, and the egress node may only represent a shared instance of the individual paths. The actual route followed by each connection may in fact later diverge from a given mp2p path previously traversed.
It should be noticed that, although the routing/forwarding processes have been decoupled with introduction of MPLS, it is the forwarding element that was re-defined, yet it still relies in the use of an external routing protocol indicating what the appropriate hop sequence will be for a proper label assignment to the packets being forwarded [3]. The question now rises as to how can the current routing mechanisms be used to find an optimal solution for the configuration of the mp2p tree.

To enable efficient set-up of mp2p trees, the MPLS technology relies on the support of a routing protocol that is capable of finding explicit (strict or loose) routes before a Label Switched Path (LSP) is either established or modified. The computation of such explicit routes may become extremely complex and computationally costly by using the current protocols [12], [14]. Because no data-aggregation considerations were taken into account during the design of the current Internet infrastructure, no efficient support is readily available towards the establishing of mp2p routes.

3 The Mobile Agent Paradigm

A number of independent research efforts have been pursued so far in the field of mobile agents for telecommunications applications, with results that seem promising for future implementation [4], [15]. Mobile agents are pieces of software code, whose objective is to perform custom computation tasks in behalf of the user. Mobile agents have been efficiently deployed in cooperative routing schemes, where each agent performs a specific task in order to obtain partial results, which are in turn shared with other agents to achieve the general routing goal [20], [21], [22].

**Wave Technology.** Although initially conceived several years ago, it was until recently that the WAVE platform was actually recognized as a true emerging technology, capable of addressing a number of issues inherent to open distributed systems [16], [17]. WAVE is described as a set of defined strings representing operations, functions or data able to propagate across a communications network. Tasks such as: optimization, modelling, topology analysis, data control and management can be efficiently and asynchronously addressed by the WAVE platform in a highly parallel and distributed manner. Classical centralized data-computation is based in the sequential execution of fetched instructions, which operate over blocks of data loaded in a memory device. Such data usually stands as an abstract representation of the state of a real-world system. As part of the mobile agents’ paradigm, WAVE integrates a number of features that overcome the limitation of the centralized schemes. The WAVE code strings, or just waves, may start its algorithmic execution at any node in the network and propagate in a controlled virus-like fashion, conquering space as their code execution evolves in time [16]. During this navigation process, the “conquered” network nodes become part of a knowledge network (KN) that behaves as a true intelligent entity distributed in space. These features make WAVE a viable tool for use in telecommunications applications, specially routing.

The fact that the WAVE technology was chosen over other platforms obeys to the fact that it was indeed designed for utilization in environments with specific requirements such as those of the communication networks. Other languages have been widely used as platforms for mobile agents’ design. Java has been observed to
be the most widely used platform for the implementation of mobile agents, which is oftentimes the language of choice for Internet applications. Different behaviours of a node and various simulation scenarios can be easily simulated by means of WAVE programs, which are very compact, typically 20 to 50 times shorter than equivalent programs written in C/C++ or Java. In [18], differences and similarities were presented between both WAVE and Java platforms, and even a combination of both was foreseen as a plausible manner of achieving a robust joint platform for mobile agents. Fig. 3, borrowed from [16], shows the layering structure of the WAVE automata, while Fig. 4 shows a sample WAVE program that finds a simple shortest path tree in a network.

![Layered Structure of Wave](image)

**Fig. 3.** Layered Structure of Wave

(\#a.F=0.RP(N~,F<N,N=F,N1=P.$.F+L))

**Fig. 4.** Sample WAVE program for finding a simple shortest path tree in a network

The controlled spreading and information sharing of waves can be used to create a logical network on top of the actual communications’ network. This logical network provides a virtual system of propagation of waves created for specific purposes. Waves manage information by means of two types of variables: nodal and frontal variables. Nodal variables are local to a given node in the knowledge network, and are shared and accessible by other waves. On the other hand, frontal variables travel along with the waves to carry information required to perform computations, and are exclusive to the wave carrying it. These variables may hold raw abstract data or even ‘passive’ code, which may be activated to become lively WAVE code with a specific purpose. WAVE also provides environmental variables for further enhancing the processing capacity of the whole system.

4 Multipoint-to-Point Routing Using Wave

In this work, two implementation scenarios for mp2p routing have been considered under the WAVE platform: static and dynamic, which fulfil the assumptions and formulations made previously. Three possible types of QoS-provision are taken into account: the traditional best-effort, assured and premium services [6], [9]. The first two types are envisioned as service agreements offered by an Internet service provider.
in which a number of network resources are pre-configured and ready to honour users’ service requests. In this respect, mp2p trees could be pre-established and kept under a static configuration premise. In contrast, the premium service agreement would follow a different scheme, in which the Internet provider would commit network resources following an on-demand basis. In such case, mp2p trees can be put together as to obey a dynamic configuration premise. This means that mp2p trees can be dynamically reconfigured when users join or leave the mp2p tree connection to preserve network throughput, while also honouring the QoS agreement. The formulation of the algorithms is now explained.

4.1 Static Multipoint-to-Point Trees

The routing algorithm presented here is divided into three parts: one for finding all possible shortest path trees from ingress to egress nodes subject to the QoS constraints, another for detecting possible merge nodes, and a last one for the final determination of the mp2p tree. As a preliminary step, a set of waves is launched into the AS to create a logical KN, which will be used by forthcoming agents to navigate and discover optimal paths. It is assumed that the nodes in the Diffserv network provide the agents with the necessary information in regards to the QoS availability for individual outgoing communication links. Therefore, a QoS-KN is set-up to represent a logical virtual network that supports specific QoS needs.

In the first part of the algorithm the egress node creates a tunnel to insert agents (waves) at all other edge nodes in the network, and from there, each wave propagates individually to find the shortest path tree (SPT), all the way back to the egress node. It is assumed that each egress node has a list of all the other edge nodes in the AS. More than one SPT might be found during this process, as the original wave can actually clone itself, so that multiple copies can propagate in a parallel manner while asynchronously searching for SPTs. Two QoS metrics can be employed here: hop-count and, say, bandwidth (e.g. it could also be delay, or any other metric). Therefore, the waves navigate through a KN whose link values are based on the bandwidth metric. The discovery of the routes will only take place over the bandwidth-constraint KN. The waves follow a breadth-first evolving-spread search technique [16], providing a highly parallel and asynchronous solution. Each time an agent reaches a new node, it firstly checks to make sure no other agents from the same originating node have been already there, and then it marks the node to advertise its presence to other waves arriving afterwards. A variable, which contains the actual distance traversed so far by that wave, is also updated. If a wave encounters that another one with the same node of origin has been already there having traversed a shorter distance, the current wave dies. Otherwise the wave is clear to proceed, which causes it to clone itself with as many copies as QoS-compliant (outgoing) links for this KN are available in the current node. This has no bond with a delay-constraint metric, since the distance being compared here is directly related to the hop-count on top of the bandwidth-KN; therefore, overridden records account for smaller hop-counts brought, not arrival-time delays. This procedure is repeated throughout the network until the wave reaches the egress node. The corresponding algorithm is shown in Fig. 5.a.
To complete the SPT procedure, a second set of waves is generated. Their task is also to propagate in a flooding-like fashion through the KN, individually recording the distance traversed. Upon reaching a node, the waves are only allowed to continue execution if the hop-count previously recorded from the origin node up to the node just reached is the same as the one currently assessed. The result is that not one but all SPTs between the ingress and the egress nodes are recorded. This same procedure is true for other waves generated from distinct ingress-egress nodes participating in the mp2p tree creation.

---

**Fig. 5.** Procedure for finding all possible SPTs between a source and a destination

**Fig. 6.** The mp2p Tree Computation

---

To complete the SPT procedure, a second set of waves is generated. Their task is also to propagate in a flooding-like fashion through the KN, individually recording the distance traversed. Upon reaching a node, the waves are only allowed to continue execution if the hop-count previously recorded from the origin node up to the node just reached is the same as the one currently assessed. The result is that not one but all SPTs between the ingress and the egress nodes are recorded. This same procedure is true for other waves generated from distinct ingress-egress nodes participating in the mp2p tree creation.
One remarkable feature of this algorithm is that it prevents the creation of cycles, since waves returning to a node previously traversed will naturally bring a larger distance than that previously recorded, causing their dying. Therefore, no further actions are needed from the MPLS side to certify that no loops have occurred. The algorithm for this second SPT round is shown in Fig. 5.b.

In the second part of the procedure, a group of waves navigate through the path collected by the previous process. They set flags at the nodes traversed so that waves originating in other ingress nodes become aware of others having found paths containing mutual nodes in their routes. This collaboration scheme helps to actually determine all candidate nodes for stream merge in the final mp2p tree. Any intermediate node automatically becomes a data-merge candidate if visited more than once by waves coming from different ingress nodes during this part of the process.

In the third and final part of the mp2p creation process, waves are again generated to navigate through the SPTs previously found. These waves assign a weight to the SPT being traversed, depending on the number of candidate merge-nodes they find: the more candidate nodes a given SPT has, the more weight it earns. Upon reaching the egress node, the weight associated with its corresponding node of origin is recorded. Any subsequent wave reaching the destination node has to compare their weight to the one previously recorded. A wave carrying a higher weight overrides previous records (i.e. paths with lower weight). As a result, the egress node will contain the set of all combined SPTs with minimal hop-count from all other ingress nodes that meet the bandwidth constraint. Fig. 6 graphically shows this mp2p process. Fig. 6.a shows an example graph where mobile agents roam the network to find SPTs, while Fig. 6.b shows the final routes chosen after SPTs with higher weight have been chosen.

By having the explicit routes ready at the egress destination, this node can call upon a Label Distribution Protocol (LDP) and pass on the address of the nodes involved in individual paths to perform the actual set-up of the mp2p tree in the MPLS network in a down-stream fashion. The details of this procedure are out of the scope of this document.

4.2 Dynamic Multipoint-to-Point Trees

For the creation of dynamic paths, almost all of the previous steps from the static case are followed. The only difference is that, when a connection leaves or joins the mp2p tree, the final path can be reconfigured again to preserve network throughput (minimize resources), while maintaining the QoS guarantees. In the case of a new connection joining the tree, a group of agents is created to follow the first part of the process to determine the SPT for the new connection. After this, all ingress nodes repeat the second and third procedures, which now includes the new connection. When a connection leaves, a set of waves is generated to delete records at the nodes involved in the SPTs that correspond to the old connection leaving the tree, and the second and third procedures of the routing process are followed again to readjust the tree. No additional steps are necessary. An actual snapshot of the WAVE simulation conducted is shown in Fig. 7, which implements a network as the one presented in Fig. 2.
5 Conclusions

We have seen how the WAVE technology could be used to efficiently compute multipoint-to-point trees in a highly distributed and parallel manner. Neither complex nor expensive combinatorial-like computations are ever performed. The agents created using WAVE can be programmed to search only paths that meet the QoS requirements of the connections involved. On the other hand, since WAVE is still a fairly new paradigm, it takes time to understand its rich semantics and the various high-level functional abstractions. However, once one becomes familiar with the new paradigm and language, the reward is significant. Performance evaluation is being conducted to determine the characteristics of the data traffic generated during the execution of the route discovery algorithm. Within the Diffserv-over-MPLS scheme, WAVE could conceivably be used to find and establish static mp2p trees according to the service level agreement previously mentioned, thereby creating predetermined traffic pathways for data transfer. Alternatively, dynamic routing could also be performed to honour requests for QoS-sensitive connections should no class-of-service/MPLS-label-binding be available at the time of the service request.

WAVE is an efficient and flexible system for distributed simulation as well as global cooperative distributed processing. The Active Agent Research Group of the Internet Computing Laboratory of UBC is experimenting with a WAVE research prototype [17], [18] as well as working on the development an improved secure version with visualization tools [19], [13].

References

Dynamic Composition of Execution Environment for Adaptive Nomadic Applications

Stefano Campadello

Nokia Research Center
P.O. Box 407, FIN-00045 NOKIA GROUP, Finland
stefano.campadello@nokia.com

Abstract: Adaptation is a key word for nomadic applications, since the execution environment that a nomadic user has varies in place and time. Traditional applications are designed and optimised for a specific environment, usually with high bandwidth and high computational power, and they do not fit well in other environments. In this paper we present a solution based on a dynamic composition of the execution environment, where the adaptation is done by construction: An instance of the application is build dynamically depending on the characteristics of the device. We introduce the concept of basic modules and the role of the Personal Agent and we present an example application.

1 Introduction

The concept of Nomadic Computing [1] comes from the desire of a user to have access to the preferred computer service at anytime and from anywhere. The increasing popularity of computing devices, such as laptops or palmtops, and their increase of computational power and usability has given the users the possibility to move around the world bringing with them their own equipment. The demand to be able not only to carry the devices but also to use them during the move has been a natural consequence.

Both academic and commercial communities have acknowledged this demand. Several solutions have been proposed that usually tend to adapt existing applications to the nomadic environment but results have not always been appealing. This is due to the fact that applications are normally designed for a specific environment, for example a high bandwidth network, and they simply do not fit well in others. This is especially true when the new environment requires fundamental adjustments. Therefore the proposed solutions are not usually portable, they often suffer from severe performance degradation, and they almost always need the direct intervention of the user in the adaptation phase.

The execution environment that a nomadic user has varies in time and place. The computing power available can range from that of a smart phone up to that of a powerful desktop. In addition, the available connectivity can be anything: Nothing at all, a couple of bytes per seconds, a few kilobytes per second or megabytes per
second. Furthermore, there will be differences in the local storage capacity as well as in input and output capabilities. Despite the hardware diversity, applications should behave reasonably and in a manner as similar as possible. In other words, applications should adapt their behaviour and resource consumption. The basic principle of adaptability is simple: When the circumstances change then the behaviour of an application changes according to the desires of the user.

In this paper we propose a solution based on a dynamic composition of the execution environment. In our approach the execution environment, both software and hardware, is decomposed into its basic elements. Given a description of the application logic, the execution environment is then recomposed dynamically by an agent that mediates between the requirements of the application logic and the constrains of the device.

The rest of the paper is structured as follow: In section 2 we give an overview of the problem space of adaptation in nomadic computing. Session 3 presents the concepts of dynamic adaptation, while Session 4 shows an example implementation of the proposed architecture. Finally Section 5 concludes the paper.

2 The Problem Space

The objective we want to achieve with the dynamic composition of the execution environment is an architecture that presents the following characteristics: Device independence, platform independence, high level of abstraction, and its adaptation is transparent to the user.

Normally applications are designed for a particular environment. This makes it easy for the designer to optimise the application for the characteristics of that environment, but it makes almost impossible to adapt the same application to a different environment. In fact, Nomadic applications will run on different devices and the device handover, or the migration of an application from one device to another with different characteristics, can happen also when the application is in its active state. Our goal is to reach device independence, so that an application can be executed on a wide variety of devices.

Once active, the application will run on top of an operating system. Our goal is to reach platform independence, so that the application can run on top of different operating systems and communication protocols but maintaining the basic application logic. For instance, the application should be able to operate in Windows or Unix environment and be able to use IIOP or Java RMI as the means of communication.

A high level of abstraction is a desired characteristic of any system. This helps the designer to reuse existing solutions or to make new ones available. For this reason we will describe our solution in terms of conceptual modules.

The user should not be forced to manually adapt her application to the new environment when roaming. The ultimate desire of the user is to have the same application anywhere. Since this is not possible due to the different characteristics of the different devices, the adaptation should be transparent, so that it should occur without the user intervention. On the other hand, the user should be able to monitor the adaptation, and, if desired, to modify it.
3 Adaptation Through Dynamic Aggregation

Fig. 1 depicts how a nomadic application adapts to the existing environment. Once an application is requested to become active, the Personal Agent examines the application logic and the basic modules (both software and hardware) available in the device. It selects the most appropriate hardware modules creating an executing environment. On the top of the executed environment the selected software modules are also aggregated to create an active instance of the application.

![Diagram showing adaptation through dynamic configuration of the execution environment]

Adaptation is done by construction: The application instance is build dynamically depending on the characteristics of the device. In the following sections we describe the components of the architecture in details.

3.1 The Basic Modules

One of the components of our architecture is represented by the software and hardware basic modules. The concept of these modules derives from an observation: In a traditional environment, applications often re-implement a same sub-service, like user interfaces or messaging services, instead of reusing already existing instances. In our architecture the services are decomposed into their "smaller" components and the decomposition continues until a bottom level is reached, where further partitioning is not possible without loosing the unique characteristic of the service. In this way we create a "community" of services that inter-operate between each other.

As an example of this deconstruction, a web browser application (see Fig. 2) can be subdivided into smaller services of "communication" and "human interaction". These services can further be decomposed. For example, the "communication" service can be decomposed in a module that implements a secure socket communication in another module that implements a streaming communication, and so on.

Hardware decomposition is done in a similar manner. A desktop computer has several basic modules: the processor that offers computational services, the RAM memory and the hard disks that offer data storage services, the monitor and the speakers that provide output service and the keyboard and the mouse that implement input services.
Every basic module implements a basic service and has specific properties. This enables the adaptation by construction; the instance of the application is done by putting together the available basic modules.

3.2 Basic module communication and advertisement

In order to be able to aggregate, the basic modules need to communicate with each other. There must be a protocol so that they can offer their services and advertise the properties of their services. Furthermore, they need a way to discover which services are offered by other modules and where these other modules are located.

The problem space described here is known as "Service Advertisement and Discovery". Several solutions have been proposed that can be used. For example, if the community of modules is mostly a compound of hardware services, the use of Bluetooth[2] looks appealing. On the other hand, to manage a community of software services we find the use of Jini[3] more interesting especially if the language environment is Java, or Salutation[4] in promiscuous environments. In any case the protocol, whatever it will be, needs to have clear and open interfaces to avoid the situation, for example, where a community of modules based on Jini is not able to collaborate with a community based on Bluetooth.

3.3 Application Logic

Every application can be decomposed in two parts: The Application logic that describes what the application should do, and the state of the application. The application logic needs to be described in a standard way. In our case the application logic should describe the interactions between different modules. It is the task of the Personal Agent to choose an appropriate software module to implement the interaction requested by the application logic.
3.4 The Personal Agent

As mentioned above, the Personal Agent has the task to find the most appropriate way to implement the application logic using the available basic modules. The task requires the ability to take sophisticated decisions and to act autonomously. In this paper we do not focus on the complex algorithms that the Personal Agent needs to use. We refer the readers to the literature on Intelligent Agents. Instead, we want to focus on the main requirement that our architecture seeks from an agent platform, that is its capability to Interoperate with other platforms.

The Foundation for Intelligent Physical Agents (FIPA)[5] is an international consortium aiming to produce specifications to establish interoperability between agent platforms. We have contributed to that forum to address the adaptation of FIPA specifications to the nomadic environment [6,7,8].

A further property of the Personal Agent is that it owns the profile of the user. This means it can "a priori" configure the application following the user's desires.

4 An Example Application: Incoming News

As an example implementation of our architecture we have the following scenario depicted in Fig. 3.

![Fig. 3. The example scenario](image)

A user has a subscription to an information service. When the subject of a news item is of interest for the user, the service provider will push the piece of news to the user's device, and the item will be displayed. The user usually receives the news on her desktop at the office but she wants to receive business-related news also when travelling.
4.1 Application Logic

The application login of this scenario is quite simple. A sketch is shown in Table 1. Basically, the application needs to open a connection with the news server provider, and when a piece of news arrives, to display it on the screen.

<table>
<thead>
<tr>
<th>Table 1. Application Login</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Establish connection to server</td>
</tr>
<tr>
<td>2) Receive description of message</td>
</tr>
<tr>
<td>3) Accept/rejecting incoming message</td>
</tr>
<tr>
<td>4) Display message</td>
</tr>
</tbody>
</table>

4.2 Personal Profile

The Personal Agent owns the user profile. Therefore, it knows, for example, that business related news have high priority. It knows also that the user does not like to receive multimedia news if the display is not good enough. The user also expects to be informed about every news item, at least at the headline level.

4.3 Basic Modules

Our example scenario involves two devices: The first one is a desktop computer connected to the network through a fast connection, with a high-resolution color monitor and high computing power. The second device is a smart cellular phone, with wireless connection, low-resolution monitor, limited computing power, and restricted battery life. The basic modules we are interested in this scenario are described in Table 2 and 3.

<table>
<thead>
<tr>
<th>Table 2. Desktop Basic Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Modules</td>
</tr>
<tr>
<td>Service</td>
</tr>
<tr>
<td>Output</td>
</tr>
<tr>
<td>Output</td>
</tr>
<tr>
<td>Output</td>
</tr>
<tr>
<td>Output</td>
</tr>
<tr>
<td>Network</td>
</tr>
<tr>
<td>Processor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3. Smart Phone Basic Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Modules</td>
</tr>
<tr>
<td>Service</td>
</tr>
<tr>
<td>Output</td>
</tr>
<tr>
<td>Output</td>
</tr>
<tr>
<td>Output</td>
</tr>
<tr>
<td>Network</td>
</tr>
<tr>
<td>Processor</td>
</tr>
</tbody>
</table>
4.4 Example of Adaptation

The user enables the application while she is working at the office. The Personal Agent (PA) starts to scan the application logic and queries the community of basic modules for a Messaging service. One service implementing the SocketHiBand interface is available. The related Network service is enabled too. The PA connects to the news service provider. When the description of a new piece of news arrives, the PA analyses it and, depending on its characteristics, it requests appropriate Output service. This device has several hardware modules implementing the service, so the application is able to display almost any kind of news.

The user now decides to move from the office but she desires to keep the news application active in her Smart Phone. The PA takes care of the Device Handover. The application logic is the same but the Basic Modules are different. Therefore the application needs to be modified. The Personal Agent can complete its task in several ways. Here we describe two of them.

1. The PA requests the Messaging service that implements SocketLoBand and connects to the news server. When the description of the new news item arrives, the PA accepts only the news that can be transmitted over the wireless connection and shown by the Hardware Basic Modules of the Smart Phone. This implies, for example, the discard of all multimedia streams, images and large texts.

2. The PA communicates with another PA situated in the office device. Knowing the user profile, the local PA decides to request from the remote PA the compression of all the images, and, if possible, the creation of a news digest instead of streaming video. The remote PA will carry out these tasks using the desktop device Software Modules. The local PA will then request the Messaging service from the module that implements RMIClient. When the description of a new message arrives, the PA will request its delivery through a Remote Invocation. The modules in the office device will request the news item from the news server, will compress it and send it back as return value of the RMI call.

4.5 Comments

This scenario demonstrates the concept of dynamic composition of the execution environment. The application is constructed dynamically depending on the characteristics of the device in use. The greatest advantage is given by the use of intelligent agents. As in the proposed scenario, the exchange of information between the various Personal Agents can result in innovative solutions.

This architecture opens also several issues. One of the biggest is related to security. The possibility to combine several modules and to request services also from other devices is a powerful enhancement. However it also introduces several security threats that must be addressed.

5 Conclusions

Automatic adaptation is a key word for nomadic applications. In this paper we presented a solution based on a dynamic composition of the execution environment.
The new concepts of Basic Modules and Personal Agent are introduced, while the task of ensuring a correct adaptation is delegated to the role of the Personal Agent. An example scenario has been presented. Further work is needed to implement the proposed solution, especially regarding the several security threats that an open and distributed platform introduces.

**Acknowledge**

The author expresses his thanks to the other members of the Software laboratory in NRC, especially to professor Kimmo Raatikainen for his support and suggestions.

**References**

5. The Foundation for Intelligent Physical Agents. www.fipa.org
Network Processing of Mobile Agents,
by Mobile Agents, for Mobile Agents

Ichiro Satoh
National Institute of Informatics / Japan Science and Technology Corporation
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
Tel: +81-3-4212-2546 Fax: +81-3-3556-1916
E-mail: ichiro@nii.ac.jp

Abstract. This paper presents a framework for building network protocols for migrating mobile agents over a network. The framework allows network protocols for agent migration to be naturally implemented within mobile agents and to be constructed in a hierarchy as most data transmission protocols are. These protocols are given as mobile agents and they can transmit other mobile agents to remote hosts as first-class objects. Since they can be dynamically deployed at remote hosts by migrating the agents that carry them, these protocols can dynamically and flexibly customize network processing for agent migration according to the requirements of respective visiting agents and changes in the environments. A prototype implementation was built on a Java-based mobile agent system, and several practical protocols for agent migration were designed and implemented. The framework can make major contributions to mobile agent technology for telecommunication systems.

1 Introduction

Mobile agent technology is an emerging technology that makes it much easier to design, implement, and maintain telecommunication systems. The technology can be used in the development of various network applications. These applications often require application-specific network processing for migrating their agents over a network. For example, a typical application of the technology is network management, where an agent travels to multiple nodes in a network to observe and access the components locally. The itinerary of such a monitoring agent seriously affects the achievement and efficiency of its tasks. Moreover, a mobile agent for electronic commerce may have to be transformed into an encrypted bit stream before it can transfer itself over a network. However, existing mobile agent systems assume particular network infrastructures and cannot dynamically adapt their own network processing to the requirements of visiting agents and to changes in their environment.

This paper addresses the dynamic customization of network processing for agent migration, rather than for data transmission. I describe a new framework for dynamically deploying and changing network protocols for agent migration. My framework is based on two key ideas. The first is to apply active network technology to a network infrastructure for mobile agents. The second is to construct network protocols for agent migration within the agents themselves. That is, my mobile agent-based protocols can
transmit mobile agents as first-class objects to their destinations. Also, the protocols can be dynamically deployed by the migration of the agents that support these protocols. Therefore, my framework allows network processing for mobile agents to be adapted to the requirements of visiting agents and to changes in the environment. The framework can provide a useful testbed for implementing and evaluating different types of network processing for mobile agents.

In this paper I survey related works (Section 2), describe the design goals of my framework (Section 3), briefly review my mobile agent system, called MobileSpaces (Section 4), present several mobile agent-based protocols for agent migration (Section 5), show some real-world examples of the framework, and make some conclusions and describe research directions for developing new protocols.

2 Background

Many mobile agent systems have been developed over the last few years, for example, Aglets [10], TeleScript [16], and Voyager [11]. To my knowledge, none can dynamically extend and adapt their network processing for agent migration to the characteristics of current networks and the requirements of respective visiting agents, although mobile agents must be used in heterogeneous and dynamic network environments, for example, in personal mobile communication, wireless networks, and active networks. This is because their agent migration protocols are statically embedded inside their systems.

A mobile agent, which visits multiple hosts to perform its task, must have an application-specific itinerary. For example, a mobile agent may roam over more than one host without making any detours or may have to return to its home host after each hop instead of proceeding another destination. Also, a network-dependent itinerary is often needed for a mobile agent to travel to multiple hosts efficiently. However, it is difficult to determine such an itinerary at the time the agent is designed or instantiated because the network topology cannot always be known. Also, even if the itinerary of a mobile agent was optimized for a particular network to travel to multiple hosts efficiently, it might not be reused in another network. To overcome this problem, ADK [8] separates the travel itinerary of an agent from its behavior by building a mobile agent from a set of component categories: navigational components responsible for a travel itinerary and performer components responsible for executing one or more management tasks on each node. Aglets [10] introduces the notion of an itinerary pattern, which is similar to design patterns in software engineering, to shift the responsibility for navigation from an application-specific agent to a framework library described in [1].

Both approaches allow us to design the application-specific itinerary for an agent independent of the logical behavior of the agent, but the itinerary parts must be statically and manually embedded in the agent. Consequently, the agent cannot dynamically change its itinerary and cannot travel beyond its familiar networks.

There have been many attempts to apply mobile agent technology to the development of active networks [2, 4] because mobile agents can be considered a special case in mobile code technology, which is the basis of existing active network technologies. For example, the Grasshopper system offers an active network platform consisting of stationary and mobile agents as service entities for telecommunication. In contrast, the
framework presented in this paper applies active network technology to mobile agent technology.

I described a portable and extensible mobile agent system, MobileSpaces, in my previous paper [12]. The system serves as the basis for the framework presented in this paper. It can dynamically adapt its functions and structures to changes in the environments. Also, I presented an architecture for building adaptive protocols in [14]. While in the previous papers I did not focus on any approach to building application-specific protocols for agent migration, the goal of this paper is to design and implement a layered architecture for building and deploying configurable protocols for agent migration and present several protocols for agent migration.

3 Approach

The goal of the framework presented in this paper is to provide a self-configurable infrastructure for agents migrating over a network. This section outlines the overall architecture of the framework and describes the basic idea of network protocols based on the framework.

3.1 Mobile Agents as First-class Objects

Mobile agents are autonomous programs that can travel between different computers. In the framework presented in this paper, mobile agents are computational entities like other mobile agents. When an agent migrates, not only the code of the agent but also its state can be transferred to the destination. The framework is built on a mobile agent system, called MobileSpaces, presented in [12]. The system is characterized by two novel concepts: agent hierarchy and inter-agent migration. The former means that one mobile agent can be contained within another mobile agent. That is, mobile agents are organized in a tree structure. The latter means that each mobile agent can migrate to other mobile agents as a whole, with all its inner agents, as long as the destination agent accepts it, as shown in Fig. 1. A container agent is responsible for automatically offering its own services and resources to its inner agents, and it can subordinate its inner agents. Therefore, an agent can transmit its inner agents to another location as first-class objects [5], in the sense that mobile agents can be passed to and returned...
from other mobile agents as values. As a result, network protocols for agent migration can be implemented within mobile agents.

3.2 Layered protocols for agent migration.

Most protocols for data transmission are often arranged in a hierarchy of layers. Each layer presents an interface to the layers above it and extends services provided by the layer below it. The hierarchical structure of mobile agents enables network protocols for agent migration to be organized hierarchically. That is, each agent hierarchy consisting of mobile agent-based protocols can be viewed as a protocol stack for agent migration, as shown in Fig. 2, and agent migration in an agent hierarchy is introduced as a basic mechanism for accessing services provided by the underlying layer. Mobile agent-based protocols in the bottom layer correspond to data-link layered protocols. They are responsible for establishing point-to-point channels for agent migration between neighboring computers. The middle layer corresponds to routing protocols for agent migration and provides a mechanism to transmit mobile agents beyond the channels between directly connected nodes. The framework enables routing protocols for agent migration to be performed by mobile agents.

![Fig. 2. Architecture of mobile agent-based protocols for agent migration.](image)

4 MobileSpaces: An Extensible Mobile Agent System

This section briefly reviews MobileSpaces, which provides, in addition to mobile agent-based applications, an infrastructure for building and executing mobile agents for network processing. MobileSpaces is built on a Java virtual machine and mobile agents are given as Java objects. Its architecture is designed based on a micro-kernel architecture and consists of two parts: a core system and higher-level components. The former offers only minimal and common functions, independent of the underlying environment. The latter is a collection of higher-level components outside the core system that provide other functions, including agent migration over a network, which may depend on the surrounding environment.

4.1 Core System

Each core system is made as small as possible for portability. It has only three functions.
Agent Hierarchy Management: Each core system corresponds to the root node of an agent hierarchy, which is maintained as a tree structure in which each node contains a mobile agent and its attributes. Agent migration in an agent hierarchy is performed simply as a transformation of the tree structure of the hierarchy.

Agent Execution Management: Each agent can have more than one active thread under the control of the core system. The core system maintains the life-cycle state of agents. When the life-cycle state of an agent is changed, for example, at creation, termination, or migration, the core system issues certain events to invoke certain methods in the agent and its containing agents.

Agent Serialization and Security Management: The core system has a function for marshaling agents into bit streams and unmarshaling them later. The current implementation of the system uses a Java object serialization package for marshaling the states of agents, so agents are transmitted based on the notion of weak mobility [6]. The core system verifies whether a marshaled agent is valid or not to protect the system against invalid or malicious agents, by means of Java’s security mechanism.

4.2 Mobile Agent Program

Each mobile agent consists of three parts: a body program, context objects, and inner agents as shown in Fig. 3. The body program is an instance of a subclass of abstract class \texttt{Agent}. This class defines fundamental callback methods invoked when the life-cycle of a mobile agent changes due to creation, suspension, marshaling, unmarshaling, destruction etc., like the delegation event model in Aglets [10]. It also provides a command for agent migration in an agent hierarchy, written as \texttt{go(AgentURL destination)}. When an agent performs the command, it migrates itself to the destination agent specified by the argument of the command in the same agent hierarchy. An inner agent cannot access any methods defined in its container agent, including the core system. Instead, each container is equipped with a context object that offers service methods in a subclass of the \texttt{Context} class, such as the \texttt{AppletContext} class of Java’s Applet. These methods can be indirectly accessed by the inner agents of a container to get information about and interact with the environment, including the container, sibling agents, and the underlying computer system.

5 Mobile Agent-Based Protocols for Agent Migration

Since this framework can treat mobile agents as first-class objects, various types of network processing for mobile agents can be implemented as special mobile agents, called service agents, running on the core system of MobileSpaces. These service agents are hierarchically organized as a protocol stack.

- Each service agent is designed to provide its service to its inner mobile agents. Therefore, each service agent in a lower layer can be viewed as a service provider for agents in an upper layer. The movement of an agent to a service agent in a lower layer in the same agent hierarchy corresponds to the process of applying the network service of the service agent to the moving agent.
Each runtime system permits one service to be provided by one or more service agents. That is, different network protocols can be supported by different service agents. Moving agents or upper-layer protocols can dynamically select a suitable agent for their requirements and migrate their inner agents to the selected agent.

Since service agents for performing protocols are still mobile, the protocols can be dynamically deployed at hosts by migrating the agents to the hosts.

Hereafter, I present several basic protocols for agent migration. Since these protocols are given as abstract classes in the Java language, we can easily define further application-specific protocols by extending these basic protocols.

5.1 Point-To-Point Channels for Agent Migration

Agent migration between neighboring hosts can be provided by mobile agents, called transmitters. They are responsible for establishing point-to-point channels for agent migration and can automatically exchange their inner agents through their common communication protocol. After an agent arrives at a transmitter agent from an upper layer, the arriving agent indicates its final destination. The transmitter suspends the arriving agent (including its inner agents), then serializes its state and codes. Next, it sends the serialized agent to a coexisting transmitter agent located at the destination. The transmitter agent at the destination receives the data, reconstructs the agent (including its inner agents), and migrates it to the destination or specified agents for offering upper-layer protocols.

5.2 Routing Mechanisms for Agent Migration

Application-specific mobile agents often need to travel to multiple hosts to perform their tasks. However, it is difficult to determine the itinerary at the time the agent is designed or instantiated. Therefore, I introduce two approaches to determining and managing the itinerary of agents. These approaches are based on transmitter agents running on hosts and correspond to different kinds of application-specific routing protocols.
**Forwarder Agent:** The first approach provides a function similar to that of an active node (also called a programmable node) in active network technology. I introduce a service provider, called a forwarder agent, for redirecting moving agents to new destinations. Each forwarder agent holds a table describing part of the structure of the network and can be dynamically deployed at a host. When receiving agents, it can propagate certain events to its visiting agents instructing them to do something during a given time period and then redirects the agents to their destinations through point-to-point channels established among multiple hosts as shown in Fig. 4. Each forwarder agent will repeat the entire process in the same way until its visiting agents arrive at their destinations.

![Fig. 4. Routing agents for forwarding the next hosts.](image)

**Navigator Agent:** The second approach is similar to the notion of an active packet (also called a programmable capsule) in active network technology. Existing mobile agents can move from one host to another under their own control, as active packets can define their own routing. I propose a service provider, called a navigator, to convey inner agents over a network, as shown in Fig. 5. Each navigator agent is a container of other agents and travels with them in accordance with a list of hosts statically or algorithmically determined, or dynamically based on the agent’s previous computations and the current environment. That is, a navigator agent can migrate itself to the next place as a whole, with all its inner agents. Upon its arrival at the place, the navigator propagates certain events to its inner agents. After the events have been processed by the inner agents, the navigator continues with its itinerary.

### 5.3 Protocol Distribution

Given a dynamic network infrastructure, a mechanism is needed for propagating mobile agents that support protocols to where they are needed. The current implementation of
this framework provides the following three mechanisms: (1) mobile agent-based protocols autonomously migrate to hosts at which the protocols may be needed and remain at the hosts in a decentralized manner; (2) mobile agent-based protocols are passively deployed at hosts that may require them by using forwarder agents prior to using the protocols as distributors of protocols; and (3) moving agents can carry mobile agent-based protocols inside themselves and deploy the protocols at hosts that the agents traverse. These mechanisms can improve performance in the common case of agent migration, i.e., a sequence of agents that follow the same path and require the same processing. All the mechanisms are managed by mobile agents, instead of by a runtime system. As a result, the deployment of transmitter agents must to be performed by other transmitter agents.

5.4 Current Status

The framework presented in this paper and its mobile agent-based protocols were implemented on MobileSpaces in the Java language. They can be run on any computer with a JDK 1.2-compatible Java runtime system. The framework provides several useful libraries for constructing network protocols within mobile agents. Several mobile agent-based protocols were developed, in addition to the protocols presented in the next section. They include agents for establishing channels through TCP, HTTP, andSMTP, forwarder and navigator agents for traveling among multiple hosts according to their own static routing tables and SNMP agents at each hosts. The current implementation of this framework was not built for performance. However, in order to compare two routing protocols, the forwarder agent protocol and the navigator agent protocol, I measured the per-hop latency in microseconds and the throughput of a single node in agents per second in a network consisting of eight PCs (Intel Pentium III-600 MHz with Windows 2000 and JDK 1.3) connected by 100-Mbps Ethernet via a switching hub. In both cases, I migrated a minimal-size agent that consisted of only common callback methods invoked at changes in its life-cycle state by the runtime system. The size of the moving agent was about 4 Kbytes (zip-compressed). For reference, I measured
the time of migrating the agent in an agent hierarchy and between two hosts. The time of migrating the agent in an agent hierarchy was 5 ms, including the time of checking whether the visiting agent was permitted to enter the destination agent. In this experiment, agent migration between neighboring computers was performed by using simple TCP-based transmitter agents. The per-hop latency of migrating the agent between two computers was 34 ms per hop and the throughput was 10.8 agents per second. The latency is a sum of marshaling, compression, opening a TCP connection, transmission, acknowledgment, decompression, and security verification.

The per-hop latency of migrating the agent using a simple forwarder agent running on the hosts was 38 ms per hop and the throughput was 9.2 agents per second. The forwarder agent determines the host that its inner agents will visit at the next hop according to its own routing table. In contrast, the per-hop latency of migrating the agent using a simple navigator agent running on the computers was 42 ms per hop and the throughput was 8.3 agents per second. The navigator agent migrated itself and its inner agents to the hosts sequentially by incorporating itself into a transmitter agent.

In this preliminary experiment, the forwarder protocol was better than the navigator protocol, because the latter protocol had to migrate not only the target agent but also the protocol itself. Also, in both protocols when more than one agent was migrated on a network, the congestion of each computer was occasionally unbalanced, because these agent-based protocols are performed asynchronously. All the above results were measured in a trial without any performance optimization and are thus difficult to evaluate. However, the overhead of the mobile agent-based protocols in terms of the latency of each agent migration was reasonable for a high-level prototype of application-specific protocols for agent migration, rather than for data communication. The throughput of each agent migration was limited by the security mechanism of the MobileSpaces system rather than by the protocols. I believe that the current throughputs are fast enough for the deployment of mobile agent-based applications.

6 Examples

This section describes three practical examples of this framework to demonstrate how it can be used.

6.1 Network Management System

A typical application of mobile agents is as a monitoring system for network management. A discussion on the suitability of mobile agents in network management can be found in [3, 9]. A system for locally monitoring equipment located at hosts in more than one network was constructed. The system consists of a monitor agent and navigator agents. The monitor agent has no mechanism for its own itinerary and thus is not dependent on any network. In contrast, each navigator agent is optimized for navigating in each of the networks and is responsible for periodically traveling among hosts in its networks. When a monitoring agent is preparing to monitor a network, it enters a navigator agent designed for that network. The navigator then generates an efficient travel plan to visit certain hosts in the network. Next, it migrates itself and the monitoring
agent to the hosts sequentially. When it arrives at each destination, it dispatches certain events to its inner agents.

6.2 Locating Mobile Agents

When an agent wants to interact with another agent, it must know the current location of the target agent. Therefore, a mechanism for tracking a moving agent is needed. An extension of the forwarder agent approach presented in the previous section offers such a mechanism, as shown in Fig. 6. Just before an agent moves into another agent, it creates and leaves a forwarder agent behind. The forwarder agent inherits the name of the moving agent and transfers its visiting agent to the new location of the moving agent. Therefore, when an agent wants to migrate to another agent that has moved elsewhere, it can migrate into the forwarder agent instead of the target agent. The forwarder agent then automatically transfers it to the current location of the target agent. Several schemes for effectively locating mobile agents have been explored in the field of process/object migration in distributed operating systems. Forwarder agents can easily support most of these schemes because they are programmable entities and can flexibly negotiate with each other through their own protocols.

![Fig. 6. Locating agents to locate moving agents.](image)

6.3 Agent Migration in Mobile Computing

Mobile agent technology has the potential to mask disconnections in some cases. This is because once a mobile agent is completely transferred to a new location, the agent can continue its execution at the new location, even when the new location is disconnected from the source location. However, the technology cannot often solve network failures in the process of agent migration. That is, agents can be migrated from the source to the destination when all the links from the source to the destination are established at the same time. However, mobile computers do not have a permanent connection to a network and are often disconnected for long periods of time. When a mobile agent on a mobile computer wants to move to another mobile computer through a local-area network, both computers must be connected to the network at the same time.

To overcome this problem, relay agents are constructed by extending the forwarder agent approach to the notion of store-and-forward migration, as shown in Fig. 7. This notion is similar to the process of transmitting electronic mail by using SMTP. When
an agent requests a relay agent on the source host to migrate to its destination, the relay agent makes an effort to transmit the moving agent to the destination through transmitter agents. If the destination is not reachable, the relay agent automatically stores the moving agent in its queue and then periodically tries to transmit the waiting agent to either the destination or a reachable intermediate host as close to the destination as possible. The relay agent to which the moving agent is transferred will repeat the process in the same way until the agent arrives at the destination. When the next host on the route to the destination is disconnected, the moving agent is stored in its current place until the host is reconnected. When a mobile computer is attached to a network, its relay agent multicasts a message to relay agents on other connected computers. After receiving a reply message from the relay agents at the destinations of agents stored in its queue, the relay agent tries to transfer those agents to their destinations.

7 Conclusion

This paper described a framework for building a self-configurable infrastructure for agent migration. This framework provides a layered architecture for network protocols for migrating agents and allows these protocols to be naturally implemented within mobile agents. Therefore, network processing for mobile agents can be dynamically added to and removed from remote hosts by migrating corresponding agents according to the requirements of respective visiting agents and changes in the network environment. I developed several mobile agent-based protocols, for example, point-to-point channels among neighboring hosts, and application-specific routing protocols for migrating agents among multiple nodes. A prototype implementation of the framework built on a Java-based mobile agent system called MobileSpaces was carried out. The framework can greatly simplify the development of active network technology [15]. This is because mobile agents are introduced as the only constituent of this framework and thus algorithms and protocols for active networks can be constructed and reused through a single programmable abstraction for composition and refinement of mobile agents.

Finally, I would like to mention some future research directions. The performance of the current implementation is not yet satisfactory and thus further measurements and

![Fig. 7. Relay agent for tolerant network disconnection.](image-url)
optimization are needed. I intend to focus on developing other protocols in addition to the examples presented in this paper. Also, my protocols are not always dependent on my framework and thus should be applied to other active network infrastructures.

References

Design of a Mobile Agent-Based Workflow Management System

Young-Ho Suh and Han Namgoong

Computer & Software Technology Lab.
Electronics and Telecommunication Research Institute(ETRI), Taejon, Korea
{yhsuh,ngohan}@etri.re.kr

Jeong-Joon Yoo and Dong-Ik Lee

Department of Information and Communications
Kwang-Ju Institute of Science and Technology(K-JIST), Kwangju, Korea
{jjyoo,dilee}@csrl.kjist.ac.kr

Abstract. This paper deals with several architectural issues on a mobile agent-based workflow management system(WFMS). We mainly focus on performance and scalability issues among various architectural issues. We point out three major design issues that are indispensable for designing a mobile agent-based WFMS and find solutions for the issues. We propose an efficient design strategy based on the solutions, i.e. a mobile agent-based '2-tier distributed workflow server architecture', 'process execution structure through hierarchical delegation' and 'introduction of a non-trivial delegation model'. We also present both a mobile agent based 3-tier run-time architecture and a process execution scenario, which are established according to the proposed strategy. Finally, we show the effectiveness of the proposed method by evaluating performance and scalability through GSPN simulation.

1 Introduction

A workflow management system, in short a WFMS, is a system that defines, creates and manages the execution of workflows through one or more workflow engines which interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications [1]. In order not only to coordinate and streamline business processes but also to facilitate the integration of all the information resources, a WFMS is required for being deployed as the backbone of information processing technology of an enterprise [2].

Most existing workflow-runtime systems are based on the client-server model and they are centralized in the sense that a single workflow engine handles one or more entire process executions. Unfortunately, this centralized architecture cannot support reliable and consistent process execution with acceptable failure resiliency, perform-
The use of mobile agents has been proposed for a WFMS. Mobile agents refer to self-contained and identifiable programs that can migrate over heterogeneous networks and can act on behalf of a user or another entity. Mobile agent technology may have certain advantages over the client-server paradigm, e.g. reduction of network traffic, support of mobile computing that has unreliable or non-permanent network connections, etc.

In this paper, we design a scalable mobile agent-based WFMS. Our architecture has 3-tiers, which is conformant to the workflow enactment model – a workflow process level, a process instance level and a task instance level. We point out three major design issues and figure out solution for each issue. Based on the solutions, we suggest a mobile agent based 3-tier run-time architecture and present a process execution scenario on top of the architecture. Finally, we show the effectiveness of the proposed 3-tier architecture by evaluating performance and scalability through GSPN simulation.

The remainder of this paper is organized as follows. In Section 2, we classify architectures of WFMSs into two categories according to the paradigm on which they are based and point out each of their limitation. In Section 3, we present three major design considerations of our strategy and propose a solution to each consideration. Then we design a new mobile agent-based WFMS on which the solutions to the three considerations are effectively reflected. In Section 4, we show ‘performance and scalability’ of the system through GSPN simulation and compare with two other approaches. Finally, Section 5 is the conclusion.

2 Related Work

2.1 Architectures based on the client-server paradigm

The early WFMS products adopted a centralized architecture with a monolithic workflow server. Those systems cannot support requirements of WFMSs in terms of performance and scalability, because there must happen bottleneck on the centralized workflow server. Therefore, to overcome limitations of the centralized architecture with a monolithic workflow server, there are two streams of researches - centralized architectures with multiple workflow servers and fully distributed architectures without any workflow server [5-9].

2.1.1 Centralized architectures with multiple workflow servers

This approach tackles the problems of performance and scalability of WFMSs by allowing multiple workflow servers which cooperate during maintaining the centralized architecture. Recent versions of FlowMark[5] and COSA[6] have suggested architectures which fall into this category but it is left open how the cooperation among servers takes place. Alonso et al. [7] suggested to resolve the problem of cooperation among multiple workflow servers by the introduction of clusters. A cluster consists of multiple workflow servers that execute the same workflows using a com-
mon database. Then, load can be distributed among the servers but the common database is still a potential bottleneck. Moreover, there are no hints how to build up the clusters in a given enterprise environment, i.e. how to place workflow data and how to schedule the execution of workflows. In spite that there are some variations of implementations, this approach can support limited degree of performance and scalability since additional workflow servers get started if loads increase. However, this approach has disadvantages that there must be provided an efficient cooperation mechanism among multiple workflow servers and there is overheads for the cooperation.

2.1.2 Fully distributed architectures

This approach solves the scalability problem of the centralized architecture by introducing a fully distributed architecture. There is neither a central workflow server nor a central database. IBM's Exotica project[8] suggested a completely distributed architecture in which every node is fully autonomous. Schema information about workflows types, called process types, is replicated to each node. Instance information is localized to a single node. Nodes communicate using persistent queues. Miller et al. [9] present a fully distributed architecture based on CORBA services as communication facilities. In any case, this approach can provide performance and scalability through adopting a completely distributed architecture in which workflow schema is replicated in all nodes and workflow instances are executed autonomously in every node, interchanging run-time information of workflow instances with others through messaging system or CORBA services. However, this approach suffers from severe shortcomings due to the lack of a centralized view, that is the execution of a workflow cannot be monitored and the expensive costs [10].

2.2 Architectures based on the mobile agent paradigm

The fundamental characteristics of mobile agents make a mobile agent-based WFMS scalable inherently. Each business process instance can be handled by an agent. An agent consists of a process-specific code and data; there is no need to access the central database server at every step. Thus controls and workloads can be naturally distributed throughout the entire system rather than concentrated on workflow servers. Therefore, this approach tries to overcome limitation of the centralized architecture by exploiting the potential advantages of the mobile agent paradigm. DartFlow system [4] corresponds to this approach.

2.2.1 Fundamental characteristics of mobile agents

The followings are fundamental characteristics of mobile agents [11]:

- **Delegation**: Users or other programs delegate tasks to agents and vest with authority to act on their behalf.
- **Autonomy**: An agent can make its own decisions based on the goals, preferences and policies.
- **Social ability**: Agents have the ability to interact with their peers, with the environment and with their owners.
- **Flexibility**: Agents do not assume fixed roles; they may act like clients, servers and observers, depending on their current needs.
• Mobility: Agents can move across heterogeneous computer networks to accomplish assigned tasks.

2.2.2 DartFlow system

DartFlow dealt with not only transaction-related properties of a mobile agent-based WFMS such as concurrence, availability, performance, and scalability, but also issues being inherent in the nature of WFMSs such as extendibility, flexible organization structure, and dynamic reconfiguration. Actually, from the scalability point of view, since a process agent can autonomously perform all the tasks received from the organization server in the process instance initiation phase, scalability can be naturally provided through the asynchronous nature of process execution. However, a process agent must inform the worklist server completion of each task in order for the worklist server to be able to update front-end. Therefore, if there are many process instances executed in parallel, the worklist server becomes a potential bottleneck. In addition, since a process agent is responsible for performing an entire process, performance degradation is caused by the control and migration overhead especially when the process is a set of many tasks. Moreover, they did not mention agent’s location management mechanism that is tightly related to performance and scalability of mobile agent-based systems. Namely, in spite that execution scenario of mobile agents, not to mention the entire architecture, must be properly optimized so that the potential benefits of mobile agent paradigm can be realized or reflected on the system, those considerations have not been taken into accounts.

3 Design of a mobile agent-based WFMS

We aim at designing a mobile agent-based WFMS that supports high performance and scalability. In order to maximize the performance and scalability of a mobile agent-based WFMS, the execution scenarios as well as the run-time architecture must be strategically designed so that the fundamental characteristics of mobile agents are effectively exploited. From this point of view, we consider the following three issues in our design strategy:

− How can we minimize the overhead, which is caused by co-operations among servers and corresponds to a shortcoming of a centralized architecture with multiple servers, by exploiting the fundamental characteristics of mobile agents?
− What are important considerations in the mobile agent system level in order to exert potential advantages of mobile agent paradigm?
− What is an efficient strategy to optimize the execution structure of workflow processes through delegation to mobile agents?

In this section, we propose solutions for above issues and design a mobile agent-based WFMS architecture on which those solutions are properly reflected.
3.1 Design issues and Solutions

3.1.1 Co-operations among Multiple Servers based on Mobile Agents

1. Problem - Overhead caused by cooperation among multiple servers is a shortcoming of a centralized architecture with multiple servers based on the client-server paradigm.

2. Solution - In designing a multiple server-architecture based on the mobile agent paradigm, we can consider ‘mobile agent-based 2-tier distributed workflow server architecture’. As shown in Fig. 1(a), the client-server based approach makes a workflow server expanded to multiple servers by simple replication and the resulting multiple servers work together through synchronization. In Fig. 1(b), when the workflow coordinator generates a proxy agent, the agent moves to the second layer and executes a delegated workflow instance autonomously, whose execution is managed by the coordinator located in the first layer. Therefore, cooperation among multiple workflow engines is not required. It is possible for a proxy agent to move to another workflow engine for dynamic load-balancing, but even then, cooperation among engines are not necessary because all runtime information of a workflow process is carried along with the agent.

3.1.2 Mobile Agent System Architecture Level

1. Problems - To guarantee ‘autonomous mobility’ of mobile agents, mobile agent systems must provide location/naming service, so that agents can communicate with others or can be remotely managed by the owners. In order to provide location-independent name resolution scheme, naming service is required to map a symbolic name to the current location of the agent. However, the naming service may be potential bottleneck in mobile agent systems – this may be unacceptable in such systems that a huge number of agents are executed in parallel.

2. Solution - Instead that proxy agents migrate over task performers to execute the process instance delegated by the workflow coordinator, they create sub-agents with the help of a workflow engine and delegate the execution of the process instance to the sub-agent. Sub-agents move over the task performers and execute process instances. They progressively accomplish assigned tasks by migrating one after another. Before each migration, they notify the completion of the task to

![Fig. 1. Mobile agent-based 2-tier distributed workflow server](image-url)
the proxy agent along with the location information of the next hop so that the proxy agent can keep track of their current location as shown in Fig. 2.

3.1.3 Delegation model

1. Problem - Agent migration overhead is another source of performance degradation. It is worthy noting that marshalling and unmarshalling occupy almost 90% of a single migration and increase proportional to the size of an agent[12]. Considering this, there must be provided a certain ‘process decomposition policy’ to reduce the migration overhead, more specifically overhead for marshalling and unmarshalling. However it is beyond scope of the paper to find the optimum solution. Here we propose one non-trivial division method.

2. Solution - There are two trivial delegation models. One is ‘minimum delegation model’ where one sub-agent is created for executing each unit task defined in the process as shown in Fig. 3(a). The other is ‘maximum delegation model’ where the execution of a process instance is entirely delegated to one sub-agent as shown in Fig. 3(b). In this paper, we adopt these two trivial delegation models as references.

We try to enhance performance by proposing a comparatively good delegation model that is a hybrid model of the maximum and minimum delegation models.
• The new delegation model: If the given workflow process does not contain any AND-split path, our model is just the same as the minimum delegation model. Where an AND-split path is defined as a sequence of tasks from a AND-split point to the corresponding AND-join point. Otherwise, our model is as follows;
  – Initialization: Find every AND-split path that is not included in any other AND-split paths and define it as a sub-process. For the remaining parts define each unit task as a sub-process.
  – Algorithm
1. For every sub-process containing one or more AND/OR-split paths, find every AND/OR-split path which is not included in any other AND/OR-split paths and defined it as a sub-process. The remaining parts are decomposed into sub-processes which are delimited by the AND/OR-splitting/joining points.
2. Iterate above decomposition until there does not exist any sub-process containing one or more AND/OR-split paths.

If a workflow process is given as shown in Fig. 4, in the initialization phase. It is decomposed into 12 sub-processes(dotted squares). Among them, for the two sub-processes containing AND/OR-split paths the algorithm further decomposes both of them into three sub-processes(solid squares) respectively. Finally, the workflow process is decomposed into a co-ordinated(parallel/serial) set of 16 sub-processes.

3.2 3-tier run-time architecture
The architecture consists of following components:
• Process repository
  – The schema information of all workflow is stored in forms of process template which corresponds to the intermediate data for creating proxy agents
• Workflow coordinator
  – Provides interface to workflow clients, monitoring and administrating tools.
  – Initiates process instances requested by users through creating proxy agents and dispatching them to workflow engines.
  – Provides control over tier 2 by containing primitives to communicate with or manage and monitor the proxy agents.
Workflow engine
- Provides execution environment in which a number of proxy agents can be executed concurrently.
- Supports execution of proxy agents by providing core module for execution of process instances such as scheduling module and recovery module.
- Initiates sub-process instances, according to requests from proxy agents, by creating sub-agents and dispatching them to corresponding task performers.
- Keeps primitives to communicate with, manage and monitor own sub-agents.

Task performer node
- Provides tools and interfaces for task performer to perform task instances.
- Provides execution environments for sub-agents and worklist handler-agents.

Proxy agent
- Represents a process instance and encapsulates run-time data.
- Creates and schedules sub-agents according to the given decomposition policy by requesting to the scheduler module of the workflow engine.
- Manages location information of sub-agents they created.

Sub agent
- Represents a sub-process instance and encapsulates the run-time data.
- Progressively performs all the task instances in the sub-process by interacting with task performers through the mediation of worklist-handler agent autonomously migrating one task performer node to another.

Worklist handler - agent
- Mediates the interaction between sub-agents and task performers.

4 Experimental Result

In this section we evaluate performance and scalability of our design strategy - introduction of the mobile agent-based 3-tier WFMS architecture and the delegation model. We compare our system with other two architectures — a centralized architecture with multiple workflow servers based on the client-server paradigm and a cen-
tralized architecture with a monolithic workflow server based on the mobile agent paradigm. We adopt the workflow instance shown in Fig. 4 as the target. And we use generalized stochastic Petri net model for the simulation[14]. Due to the lack of space, we omit all the details of modeling here and only show the result that is shown in Fig. 6. In Fig. 6, horizontal and vertical axes represent the number of instances and elapsed time to complete the workflow instances, respectively. Thus, in the graph, scalability is considered as oblique of each case, while performance corresponds to absolute elapsed time to complete a fixed number of workflow instances. From the observation we can easily make sure the effect of the proposed design strategy in both performance and scalability.

5 Conclusion

In this paper, we proposed a design strategy for mobile agent-based WFMSs. By hierarchical distribution of control in run-time architecture as well as system-level architecture, potential advantages of a mobile agent in WFMSs are realized in terms of scalability and performance. Namely, the mobility and autonomy characteristics endow a WFMS with flexibility and load-balancing capability and makes possible flexible execution mechanism. Further a central location server that is another potential bottleneck of a mobile agent-based system can be successfully eliminated by the hierarchical distribution of run-time architecture.

Further we showed the effectiveness of the proposed strategy by GSPN simulation. Simulation results show that our approach is better than the two existing approaches in the senses of performance and scalability. Besides, since our approach is inherently based on the mobile agent paradigm, it can satisfy other various requirements of WFMSs such as reliability, adaptability and dynamic reconfiguration.

Various issues related to the WFMS level such as ‘optimal delegation model’, ‘concurrency control’ and ‘recovery’ were not considered in the paper. Those issues come under the scheduling problem in mobile agent-based workflow management and we are currently investigating those issues.

![Fig. 6. Performance and Scalability of the three approaches](image-url)
References

Towards an Agent-based Distributed Hierarchical Network Management System for All-Optical Networks

Bilal Khan\textsuperscript{1,2}, Dardo D. Kleiner\textsuperscript{1,3}, and David Talmage\textsuperscript{1,2}

\textsuperscript{1} Center for Computational Science, U.S. Naval Research Laboratory, Washington DC 20375, USA.
\textsuperscript{2} Advanced Engineering Sciences, ITT Industries, Alexandria VA 22303, USA,
\textsuperscript{3} CIPS Corporation, Rockville MD 20852, USA

\textbf{Abstract.} This paper describes the design and implementation of Optiprism, an agent-based network management system (NMS) providing configuration and fault management services for all-optical networks. Optiprism is designed to support (1) a scalable architecture consisting of a distributed hierarchy of software agents, or managers (2) the ability to alter the hierarchy as the network evolves by adding, removing or upgrading managers (3) reorganization of physical deployment for better responsiveness (4) an innovative browser agent providing scalable end-user interaction with the distributed NMS.

1 Introduction

Traditional network management software implementations have used centralized paradigms based on SNMPv1 or SNMPv2c, or weakly distributed hierarchical paradigms based on SNMPv2, RMON, CMIP, or CMIP derivatives such as TMN [17, p. 5]. While these approaches are feasible in small networks, their communication costs grow linearly with the number of devices [24, p. 4]. Wavelength division multiplexing (WDM) networks present additional difficulties since the central problem of routing and wavelength assignment (RWA) [23] is NP-complete [26] and even heuristic approaches to it are computationally expensive [4, p. 2].

An effective optical NMS must thus address the core problem of scalability. We contend that a strongly distributed deployment of a hierarchy of cooperating intelligent mobile agents [17, p. 9] or managers would yield significantly reduced processing requirements at the client-side.

The architecture of our NMS is inspired by theories of organizational hierarchies, as developed in the works of J. R. Galbraith [10], Mount and Reiter [19], Radner [22], Patrick and Dewatripont [20] and others. We draw upon the compelling analogies comparing distributed computer systems with distributed human organizations, as presented by Fox [9], et al. In particular, our NMS employs the idea of vertical integration within a hierarchy of managers: state information is condensed and flows recursively upwards at each level of the hierarchy.
to facilitate the analysis of state and the making of decisions. This process can result in actions which are executed by a recursive downward flow of subtasks to subordinates. The two flows facilitate both decentralized decision-making and decentralized information processing, respectively, in a manner described by Van Zandt [25, pp. 1]. We draw on Galbraith’s mechanistic model of organizational design theory, incorporating the strategy of permitting lateral relationships between managers across groups. This model of planning achieves integrated action and reduces the need of continuous communication between interdependent sub-units. Within the NMS, control operations, such as lightpath provisioning, are issued to the high-level managers, who then compute routes and delegate partitioned connection requests to their subordinate managers. Monitoring of alarms and alerts operates in the reverse direction: subordinate managers report fault conditions to their supervisor. Depending on the task, the management application communicates with some subset of the managers to monitor and manipulate the network. The next sections describe the design and implementation of the Optiprism network management system.

2 Design

In designing Optiprism, we adopted a distributed architecture because it enabled us to meet four important objectives. The most critical is scalability. In large networks, the processing of management requests (e.g. route selection) presents computational burdens that would ultimately choke a centralized NMS. In contrast, a distributed architecture can amortize this computational overhead against a set of processes distributed throughout the computational environment [12, p. 1]. Second, a distributed architecture is maintainable because it is easier to augment as the network grows. Third, a distributed architecture permits computations to be closer to information sources, reducing latency and total control traffic [7] [14], thereby yielding better responsiveness. This benefit is amplified if the architecture supports dynamic re-distribution of managers, since then the NMS can adapt to circumvent computation and communication hot-spots in its environment [13, p. 32]. Finally, adopting a distributed architecture makes it possible to develop end-user management applications which exhibit scalable interaction, i.e. applications that interact with only a scalable subset of the NMS at a given time. We now describe how the design of Optiprism strives to meet these objectives.

2.1 Scalability

An effective optical NMS must able to coordinate the control planes of hundreds of optical switches. This objective led to the choice of a hierarchical architecture. In Optiprism, each manager can be a supervisor, composed of several subordinate managers. Conversely, each manager—with the exception of a unique “root”—is subordinate to some supervisor. In a supervisory role, each manager provides an
interface to the services it can implement using the functionality of its subordinates. Two managers are called peers if they have the same supervisor.

Complications arising from this design choice include: (i) higher level managers may experience greater load and (ii) failures at higher levels may have non-local negative side-effects on the NMS. Presently these concerns are addressed by assigning high-level managers to more reliable machines that have larger memory and processing power. We are investigating the possibility of addressing both issues through replication and clustering of managers.

2.2 Maintainability

The NMS architecture should be easy to alter as the network evolves. In a hierarchical NMS, this would be achieved by addition and removal of managers, and by restructuring of the hierarchy. Adding new hardware to the NMS domain should require little more than inserting a new specialized subordinate into the hierarchy. Let us see how Optiprism achieves this.

In Optiprism, there are three types of managers:

1. *Element managers* exist at the lowest level of the manager hierarchy. Each manager controls and monitors a physical device via specialized communication protocols.

2. *Subnet managers* delegate to and aggregate from lower level managers.

These two types of managers expose a command interface to the next higher level and a notification interface to the next lower level. All command and notification interfaces are functionally identical, regardless of the manager's level.

Making all subnet and element managers indistinguishable makes it possible to add, remove and splice managers into an existing Optiprism hierarchy at run-time. It has also yielded benefits of simplicity in their implementation and interactions, while providing encapsulation at the manager level. Subnet managers are truly “virtual optical switches”.

One issue with this approach is that element managers for new devices must adhere to a specification representing the least common denominator of the functionality of all devices. As vendors adopt standards for optical network provisioning and management, this penalty will be alleviated. An agent-based attempt at such standardization is [8] by FIPA.

Physical network topology is reflected by deployment of:

3. *Link managers*, each of which represent a physical connection between two elements/subnets.

Subnet managers determine their internal topology (i.e. the connectivity among their subordinate subnets/elements) by consulting subordinate link managers. In addition, they discover connectivity with peer subnets/elements by consulting their peer link managers. In the terminology of [5], all Optiprism managers can be considered *netlets* because they have a persistent process-based life-cycle model [13].
2.3 Responsiveness

The performance of an agent-based NMS is influenced by the characteristics of both the hardware on which the agents reside and the network over which they communicate. Cost factors make it impractical to dedicate entire machines and separate networks solely for the NMS. On the other hand, permitting managers to mingle with external processes on multi-purpose machines means that the system needs to sense fluctuations in performance characteristics and act to minimize impact on the NMS. This requirement underscores the need to support process mobility [13, pp. 26-33] as a core feature of the NMS. One drawback of allowing managers to be mobile is the added complexity of inter-manager communication: managers need to communicate with each other reliably despite their ability to move. Another complexity introduced is that the NMS must collect and provide sufficient information, from which decisions about manager migration can be made. Section 3.5 describes how Optiprism addresses some of these concerns.

2.4 Scalable Interaction

An NMS must provide an application for network administrators to access network management services. This application needs to communicate with the NMS’s managers so as to obtain information about the state of the network and the range of commands that may be initiated. This information would then be used to populate the application’s user-interface. Scalability dictates that the application cannot expect to communicate simultaneously with all running managers at any time.

Optiprism provides a browser agent as a scalable solution to user interaction with a large hierarchical NMS. This agent is a leaf in the hierarchy of managers and may only communicate with managers that are visible from it. This set is defined to be the browser’s peers, its supervisor’s peers, its supervisor’s supervisor’s peers, and so on up to a configurable number of levels that we call its horizon. Visibility ensures a “graceful degradation of resolution” which provides the administrator with full access to parts of the network “near” the task at hand, while still maintaining a perspective on the “bigger picture”. In the language of organizational design theory, the browser agent is considered to be a “consultant” whose position within the management hierarchy can be changed at will. This browser can establish lateral relationships with a limited set of managers relevant to its current position in the tree, and can accomplish tasks by issuing requests to them.

An administrator can change the browser agent’s location within the hierarchy in one of two ways: (i) promotion causes it to become a peer of its supervisor; (ii) demotion causes it to become the subordinate of one of its peers. This logical navigation of the browser agent causes its set of visible managers to change in a manner that corresponds to (i) zooming out and (ii) zooming in on particular regions of the network. Many browser agents can be instantiated simultaneously, to provide management from various vantage points in the hierarchy.
3 Implementation

Optiprism is implemented using a Java-based multi-agent framework called CHIME (Cellular Hierarchical Information Modeling Environment [16]), developed at the Naval Research Laboratory. Like other agent frameworks [6, 18, 27], it provides an execution environment for mobile agent code. This execution environment is called a depot. Every machine that is part of CHIME runs a depot. CHIME also provides a component API for agent development similar to the Java Agent Specification [1]. Notable differences between CHIME and prior frameworks include (i) intrinsic support for agent hierarchy, (ii) support for logical navigation, and (iii) enforcement of the visibility constraints (as presented in section 2.4). A CHIME agent may interact with the depot in which it resides and request (i) migration to a different depot, (ii) logical navigation via promotion or demotion, or (iii) a structured directory of visible agents. Optiprism managers and browsers are derived from CHIME’s agent classes and thus inherit the same capabilities.

3.1 Installing Optiprism

Optiprism has been deployed and tested on the Multi-wavelength Optical Network\(^1\) (MONET) switches [2] of the Advanced Technology Demonstration Network (ATDnet). ATDnet presently consists of six sites connected in the dual-homed multi-ring topology [21] (see top left of figure 1). Two of the sites (NRL and NSA) have Wavelength Selective Cross-Connect (WSXC) switches while the remaining four have Wavelength Add/Drop Multiplexer (WADM) units. Each WSXC supports four transport interfaces (TI). Each TI carries eight wavelengths using wavelength division multiplexing (WDM). The WADM units support two similar TIs. Each network element has several single-wavelength client interfaces (CCI) where the optical signal enters and exits the WDM layer.

In general, to install an Optiprism system, the network topology is determined by a network administrator, who partitions it hierarchically by assigning an Optiprism address to each network element and indicating link endpoints. Each address is a dotted sequence of unique names. An installer utility takes this description and instantiates a corresponding hierarchy of element, link, and subnet managers, distributing these in available depots. Each element manager immediately initiates a session with its corresponding physical device. The manager then uses this session for transmitting commands and receiving notifications from the device. Figure 1 shows the hierarchy for ATDnet.

3.2 Management Subsystems

The OSI management model categorizes network management into several functional areas. Optiprism presently addresses two areas needed in the ATDnet research environment: (i) Configuration management (CM), which addresses the

\(^1\) MONET is sponsored by the Defense Advanced Research Project Agency (DARPA)
problem of lightpath provisioning, and (ii) Fault management (FM), which enables monitoring of hardware alarms and alerts. Each functional area is embodied in a management subsystem, and a manager is then composed of a set of subsystems. Presently Optiprism subnet and element managers contain CM and FM subsystems. In the future, performance and security management subsystems will be supported.

Communication between managers takes place via delegation agents, or deglets (see [5]). A deglet is a lightweight agent with a transient task-based life-cycle model [13]. Optiprism defines two classes of deglets: downward flow ing control deglets and upward flowing monitoring deglets.

When a subnet manager receives a request, it formulates a set of subtasks for its subordinates. Each subtask is transported to a subordinate by a control deglet. Upon reaching its target manager, each control deglet attempts to perform the intended subtask. The deglet then encapsulates a report of the side effects and carries this back to the initiating manager. When all the deglets have returned, the manager aggregates the reports from below into a report for the original request. Collectively, control deglets are referred to as control flow.

A manager may send asynchronous notifications to its supervisor by using monitoring deglets. Monitoring deglets encapsulate information about changes in the beliefs [11] of their sender. Upon reaching its target supervisor, each monitoring deglet attempts to notify the supervisor of the change in the subordinate’s beliefs. The deglet then carries an acknowledgment of this notification back to the originating manager. Collectively, monitoring deglets are referred to as monitoring flow.
3.3 Configuration Management

To illustrate the operation of control and monitoring deglets, we describe how the connection management subsystem (CM) provides support for lightpath provisioning. The procedure for handling teardown requests is similar but simpler.

**CM Monitoring Flow** CM monitoring flow takes the form of *CAT-Status* deglets. These contain a Connection Availability Table (CAT) which describes the availability of routes across a subnet/element. At the element level, the CAT is the complement of the fabric table modulo the wavelength conversion capabilities of the device. At higher levels, each subnet manager generates its own CAT by aggregating the information from the CATs of its subordinates as follows.

Each CM periodically obtains a CAT from each of its subordinates. The CM maintains two graphs: (i) a *compressed graph* that has one vertex for each of its subnet/element subordinates and one edge for each of its link subordinates, and (ii) an *exploded graph* derived from the compressed graph by replacing each link with a set of parallel edges (one per wavelength) and replacing each vertex with the CAT obtained from the corresponding subordinate. Figure 2 depicts the relationship between the compressed and exploded graphs. A vertex in the exploded graph corresponds to a *particular wavelength* on an interface advertised by some subordinate. The CM considers each pair of wavelengths $\lambda_1, \lambda_2$ where either (i) $\lambda_1$ is a wavelength on a border input transport interface (TI) and $\lambda_2$ is a wavelength on a border output TI, or (ii) $\lambda_1$ is a wavelength on an input compliant client interface (CCI) and $\lambda_2$ is a wavelength on a border output TI, or (iii) $\lambda_1$ is a wavelength on a border input TI and $\lambda_2$ is a wavelength on an output CCI. For each such pair, the CM uses its exploded graph to compute a route between the corresponding vertices. If a route is found, the CM makes an entry in its *own CAT*. Once the CM has considered all such pairs $\lambda_1, \lambda_2$, it sends the constructed CAT upwards to its supervisor. This procedure recurses upwards.

Several schemes are used to speed up CAT aggregation. To reduce the number of computations required in CAT aggregation, access points (CCIs) are grouped based on their connectivity within that subnet. The precise criteria for determining “similar connectivity” is tunable, in order to obtain an acceptable trade-off between accuracy and computational cost. To reduce the frequency of CAT computation, a random sampling of CAT entries is recomputed periodically and used to estimate the likelihood that a new CAT would be “significantly different” from the one previously advertised. Whenever this likelihood exceeds a threshold, the entire CAT is recomputed. We also use techniques similar to those proposed for reducing routing traffic in optical OSPF [3].

**CM Control Flow** Lightpath provisioning is achieved by CM control flow. Requests are delivered via deglets to the highest subnet manager containing both endpoints of the desired trail. From there, requests proceed recursively in parallel
down the tree until they reach element managers, which create individual fabric connections in hardware. The trail partitioning process follows the guidelines of ITU-T G.805 [15]. To perform routing, each manager uses its exploded graph to determine a suitable path across the subnet. The path determines a set of lightpath provisioning subtasks that are then sent to appropriate subordinates via control deglets. Returning deglets indicate the success or failure of each subtask. A failure can result in a fail-fast response (i.e. rollback of any completed subtasks, and immediately report failure to the supervisor) or a reroute response (i.e. attempt to route around uncooperative subordinates).

3.4 Fault Management

The purpose of the Fault Management subsystem (FM) is to detect and diagnose network faults. We describe the roles of control and monitoring deglets in the FM.

**FM Monitoring Flow** The monitoring flow for the FM consists of fault notifications. These are encoded in Fault-Indication (FI) and Fault-Clear (FC) deglets which convey severity, location, and type of network failure. FI/FC messages propagate upwards in the tree. Intelligent filtering is performed at each level, customized to the particular monitoring characteristics desired (e.g. severity, location, type, etc). Each FM filters and aggregates fault information received from its subordinates and passes this upward to the next higher level.

**FM Control Flow** The control flow of the FM enables run-time configuration of the corresponding monitoring flow for an FM-enabled subnet or element.
manager. For example, the parameters determining the fault aggregation policy of each FM are configurable via control deglets. Similarly, control deglets are used to register Fault-Handlers inside an FM. Whenever an FM receives an FI/FC message from a subordinate, it reports this message to each registered Fault-Handler, which can then determine how to respond to the error condition.

3.5 Manager Communication & Mobility

Allowing managers to be mobile introduces complications to inter-manager communication. Optiprism addresses these issues by using CHIME’s two-layer inter-agent communication protocol stack. The Inter-Cell Transport Layer (ICTL) provides FIFO delivery between pairs of agents, and below it, the Inter-Depot Transport Layer (IDTL) provides FIFO delivery between pairs of depots. Managers communicate via ICTL messages which are encapsulated into IDTL messages during inter-depot transit. The address of the target depot is obtained by resolving the name of the destination agent using a distributed agent look-up service. Inbound messages are unpacked and delivered to their target only if the target’s name is found in the directory of local agents. Otherwise, the sending agent is blocked from further communication with the target, until its local look-up service has obtained a new binding.

Optiprism uses CHIME’s Traffic Analyzer Module (TAM) and Microbenchmark Facility (MBF) to give managers information needed to make decisions about migration. The TAM maintains statistics on round-trip latency and cumulative volume of traffic from each locally resident manager to the depots with which it communicates. The MBF takes local measurements of average CPU and memory usage. A manager may use this information to determine when to request migration, and to where. CHIME follows the paradigm of “Agent proposes, Depot disposes”. Either the source or the destination depot can reject an agent’s request to migrate. We are further investigating optimal criteria for (i) when managers should request to migrate and (ii) when depots should allow managers to migrate into or out of them.

3.6 The Management Browser

The browser communicates each visible manager $A$ by collecting a model of $A$. This model $M(A)$ is an active object created dynamically by $A$, with functionality specialized to the capabilities of the browser agent. $M(A)$ maintains a bidirectional channel to its backing manager $A$; this channel operates transparently to physical mobility of the manager.

The browser displays a window to the user and asks each collected model to render itself as a user-interface component within this window. The visual representation of each model depicts the state of its backing manager (e.g., network elements are rendered as images reflecting their operational characteristics).

The browser agent is “featureless” except for its ability to navigate within the hierarchy. As the browser is made to navigate, it updates the set of models it owns based on visibility, and refreshes the window by requesting the models to
render themselves. All other functionality comes directly from the models. This design makes it possible to perform live upgrades of manager software without altering the browser.

The user can interact with the visual representations of models to get more information or issue requests. The browser dispatches mouse clicks and key presses to the model over which they occur. The model can perform immediate action or present additional dialogs for extended input. For example, each subnet manager's model provides a dialog to select pairs of input/output connection points for trail provisioning. These models also offer extended FM information in a dialog that lists the outstanding fault conditions.

4 Conclusion

Optiprism's scalable and maintainable architecture relies on the distributed deployment of a hierarchy of cooperating intelligent manager agents. By using CHIME services, managers and browsers have access to physical mobility and logical navigation. The Optiprism browser provides a management application which supports scalable interaction with NMS services. Optiprism has been successfully deployed within the ATDnet optical network.

Enhancements to Optiprism will include (i) design and implementation of the performance and security management subsystems, (ii) devising algorithms for fast CAT aggregation within the CM subsystem, and (iii) determining effective policies for manager migration, to enable the NMS to circumvent computation and communication hot-spots in its environment.

References

Towards an Agent-based Distributed Hierarchical Network Management System

A Policy Management System for Mobile Agent-Based Services

Mouhsine LAKHDISSI, Hamid HARROUD, Ahmed KARMOUCH, and Cliff GROSSNER

Multimedia and Mobile Agent Research Lab
School of Information Technology & Engineering (SITE), University of Ottawa
161 Louis Pasteur St. Ottawa, ON, Canada K1N 6N5
{mlakhdis,hharroud,karmouch}@site.uottawa.ca

Abstract. In the Multimedia and Mobile Agent Research Laboratory an underway work is conducted toward combining management policies and Mobile Agents in the area of collaborative applications. The goal is to have a collaborative system thoroughly based on agents and highly flexible and dynamically manageable through multi-level policies. Towards this objective, we have designed a global framework to support policies. This framework is used to define, store and evaluate policies defined through multiple levels of the collaborative system. Moreover, the policies are distributed judiciously over the application. This paper describes the Policy management System we have elaborated and illustrates its integration with the V-Team system through several examples.

1 Introduction

Advances in telecommunication and information technology coupled with globalization of markets and business processes and the changing lifestyles and aspirations of the workforce have facilitated the emerging of new concept of work team structures called Virtual Teams. Virtual Teams require a supportive communication network and collaborative tool set to be effective. In the Multimedia and Mobile Agent Research Laboratory we have developed an agent-based multimedia collaborative environment, the V-Team project, to support virtual teams. The main components of V-Team are (i) recognizable team of people; (ii) a set of applications; (iii) a set of network capabilities; and (iv) a collection of rules binding these elements together such that they behave in a consistent manner and uniquely tailored to satisfy team needs.

One of the challenging issues in developing a framework to support virtual team is to define a model and architecture to capture and manage the dynamic behavior of the context associated with the virtual teams across space and time. That is the role and obligations of the team members, resource allocated to the team (e.g. system and network resources), and the tools used to achieve collaboration. To devise such a highly dynamic environment we have developed a Policy Management System (PMS)
for tracking the movements of the teams, binding and controlling access to team resources as well as adapting the resources to the virtual team needs.

In contrast with the existing policy infrastructures, our policy management system has the advantages of being application-independent and agent-based. Moreover, the policy architecture can be used to extend existing agent platforms, such as FIPA-OS, to include policy-based features. Another advantage is the introduction of distributed policy enforcement strategy.

The remainder of the paper is organized as follows. The next section provides a discussion on the definition of policies and possible levels of policies. Section 3 presents the Policy Management System with a focus on policy creation, storage and enforcement. Section 4 describes the V-Team infrastructure and illustrates how the concept of policies is applied. Section 5 discusses some implementation considerations. Section 6 highlights related work. Section 8 presents our conclusion and future work.

2 Definition and Levels of Policies

Researchers have associated various definitions with the concept of policies. The dictionary defines a policy as “a general principle or plan that guides the actions taken by a person or group”. A more technical though still general view of policies is the one provided in [1] that considers policies as “one aspect of information, which affects the behavior of objects within the system”.

In this paper we define a policy as follows: “A policy is a set of rules reflecting an overall strategy or objective, affecting the behavior of agents and thus designed to help control and administer a system”. A policy rule is a set of actions to be performed by a subject agent on a target agent providing some conditions are satisfied and/or some events are triggered.

The conditions, events and actions are all related to one or more agents of the system. Conditions are typically based on the state of an agent or a resource in the system. The events are triggered either by a time-period condition, a change in the state of an agent or as a result of an action (i.e. before/after events). The actions are merely agent’s methods invocation.

![Fig. 1. Policy Definition and Levels](image)

As depicted in Fig. 1, policies could be defined at different levels of the system. The lower level is the network level where policies provide tools to manage and control network devices (e.g. routers, switches...). At a higher level is the application.
In this level the behavior of the agents as well as their interaction with each other are monitored by means of policies.

A service is part of the application and corresponds to a functionality of the system. An example of a service is the teleconferencing system in a collaborative application. We could therefore define policies to manage a specific service. At the organization level, policies are used to express the role of each member in the team.

3 Policy Management System

3.1 Design Goals and Requirements

From our experience with the V-Team application and the use of policies in previous related work [2], we have derived a set of requirements and design goals that are used to design and implement a dynamic and adaptive Policy Management System. It is worth noting that our PMS could be seen as a policy service handler, which provides means by which an application can use, manage and enforce policies. The following are our main design requirements:

1- The initial design architecture of the Policy Management System was driven by the semantic of V-Team application. However our goal is to achieve a design that would be a trade-off between a fully application dependant architecture and multi-application-based architecture.

2- The PMS architecture should be able to provide services and tools to allow the application administrators, developers and possibly users to edit policies, and to assign these policies to the appropriate components in the application.

3- The PMS should provide the capabilities for translating policies from human understandable requirements to low-level routines when necessary and enforcing these policies at run-time.

4- Since the Policy Management System architecture is agent based, the applications that are using PMS services must also be agent-based. For instance our V-Team application (used to validate the PMS) is full agent-based system.

5- Policies may be needed at various levels of the application architecture and functionality. Hence the PMS should support different types of policies (e.g. system resources, storage, network, QoS, and security).

3.2 System Architecture

With the above requirements and design goals in mind, we have developed an agent-based architecture, which comprises several agents that work together to provide the Policy Management services to the applications. In this architecture, two agents are of particular interest, the Policy Management Agent (PMA) and the Policy Service Agent (PSA). The Policy Management Agent has the role of defining, editing, storing and assigning policies. To accomplish task, the Policy Management Agent may access the application profile stored in the Policy Information Base (PIB).

The role of the Policy Service Agent is to carry out the task of interpreting and enforcing policies. This requires a sustained communication between the policy
service agent and the application (i.e. agents to which policies are assigned). This communication is used to exchange and negotiate policy information updates that would impact the behavior of the agents representing the application at the run-time. We now describe the main components of the architecture.

**Policy Management Agent (PMA).** The purpose of the PMA is to assist the administrator of the application through the policy editing process. PMA is also responsible for detecting static conflict between policies that may occur during the editing process. Policies are stored in the Policy Information Base under the management of the PMA. The PMA is application-independent. Information about the application is stored in the application profile within the PIB. Application profile is a set of attributes (e.g. application structure and components) used by the PMA to communicate with the applications.

The PMA is a stationary agent that resides in the administrator’s site. It guides the administrator through the process of editing a policy. This agent is composed of the following components: Policy Editor, Policy Translator, Policy Conflict Detector and Policy Audit Tool.

**Policy Service Agents (PSA).** Each application is associated with a Policy Service Agent that is responsible for locating the events and conditions likely to trigger a policy. To trigger a particular policy, the PSA invokes a set of actions to be executed. For each triggered policy, PSA keeps track, in a log file, of the result of policy evaluation and enforcement.

PSA can be compared to the role of the PDP (Policy Decision Point) or the Policy Server in the IETF framework [3]. The PSA is the main component of the PMS and can be considered as the bridge between the application and the Policy Management Agent.

The Policy Manager (PM) is the key component of the PSA. Its task is to extract policies from the PIB and to coordinate their execution by the other four components of the PSA. The Event Listener (EL) is set by the PM to listen for events relevant to a policy. As soon as an event is intercepted the information is transferred to the PM for decision-making. The PM checks the policies and commands the Constraint Manager.

![Fig. 2. Global Architecture of the PMS](image-url)
to ensure that all corresponding conditions are satisfied. Once this step is performed, the PM can determine the actions to be invoked by the Action Performer. The Exception Handler looks over the outcome of the action and reacts to possible exceptions.

Policy Information Base (PIB). Policies are stored and maintained in the Policy Information Base. This database contains mainly information about the application architecture and components and the policies that are assigned to the application. The application’s information (that we call Application Profile) is retrieved at the registration phase and consists of the following:

- **Agents.** Agents composing the application defined in a hierarchical structure to highlight relationships between them.
- **Services.** Actions performed by an agent and made public for use by any entity in the system. The policies that will be defined will make use of these services.
- **Attributes.** Attributes (or variables) defining the state of an agent. This information could serve to define policy conditions and events.
- **Resources.** Hardware or software resources that an agent is managing. As shown in Fig. 2, agents monitor all resources in the application.

The administrator of the system will use the PMA to define and assign policies to the registered agents based on this information.

### 4 Managing Policies for the Virtual Team Application

We have used the Policy Management System in the V-Team application, which is a collaborative framework for virtual organization and virtual teamwork. In the following section we first give a brief introduction of the application, then we show how the integration of both the application and the policy management is orchestrated.

#### 4.1 Application Architecture and Components

The goal of the V-Team system is to develop a collaborative environment that would provide a set of team services for better collaboration between virtual team participants. It also provides facilities for managing virtual team attributes and the context customization of their collaboration. Team services include multimedia conferencing, and distance group meeting facilities.

![Fig. 3. V-Team Agent-Based Architecture](image-url)
We have designed the V-Team system as agent-based architecture. Agent technology is expected to enable rapid development of robust and reusable software [4]. Agents cooperate and communicate with each other, and have the ability to communicate and monitor the execution of an application. In order to supervise the behavior of an agent and its decision making, we assign to each agent of the system a set of policies. The use of policies can highly reduce the system complexity, while permitting an efficient control of virtual team activities.

The key component of V-Team system is that of V-Team Context Agent (VTC). VTC agent controls information about team participants’ attributes, their capabilities and roles, services to be used by all or certain team members, logical resources requested when creating the team, network services and different forms of underlying transport mechanisms. By gathering information about a virtual team and its context, the VTC agent generates a set of policies that monitor the behavior of agents representing participants in a team. Participants’ Agents (PA) representing the end-users engage into negotiation process, under the control of the VTC agent, to setup a collaborative session. The Team Participant Agent is a user interface that allows each participant to access to V-Team services according to his/her role within the team and his/her privileges.

V-Team also features the network service agent (NSA) and team control agent (TCA). NSA provides a simpler interface to network services, such as mobility management, CoS/QoS, multi-party, peer to peer, multimedia session control. It alleviates team services and participants to deal with different network services directly. NSA also permits the underlying mechanism to change transparently between, for instance, multicast and unicast as needed. NSA interfaces with different network services through wrapper agents, such as W-SIP for participants’ mobility or W-RSVP for making resource reservations.

TCA supports virtual team collaborative work sessions and controls interactions between active virtual team members. It also manages the distributed event serialization and dispatches them to each active virtual team participant, through the Team Service Agent (TSA) that establishes a secure communication among virtual team workstations.

### 4.2 V-Team and PMS integration

As shown in Fig. 4, we have applied policies throughout the V-Team system from the organizational level down to the network level. The integration of the PMS with V-Team is achieved in three steps:

- The first step consists of registering the agents of V-Team (e.g. VTC, TCA, NSA...etc) with the PSA so that we populate the application profile PIB as indicated in step 0 and step 1 of Fig. 4.
- In the second step we define various kinds of policies through the PMA (step 2-3) and we store them in the PIB.
- The third step is to observe at run time the events entailing a policy enforcement and react accordingly. Events may be automatically be intercepted by the PSA. Agents may also send events to request a policy evaluation from the PSA. Upon detection of such an event the PSA invokes the actions associated with the event or send these actions in a reply message to the agent. This is shown in step 4 and 5 a, b, c, d).
4.3 Use Cases

To illustrate the use of policies in the context of virtual teams and how they could possibly be used to define and manage the behavior of the system, we provide some multi-faceted use case that could be applied in a variety of situations and at different levels. We have adopted the notation defined in [1] for authorizations and obligations.

**Security.** A+ (and A-): Only machine with IP addresses in this range (137.122.109.1 – 137.122.109.80) can join a session. This rule applies at the network level and will be translated into the following:

```
On (join session request)
If (Agent.HostIPAddress in [137.122.109.1, 137.122.109.80])
Then NMS.acceptAccess()
Else NMS.rejectAccess()
```

**Quality of Service.** A+: The team member Joe can use audio as well as video conferencing while Bob is only allowed to use audio.

```
On (Trigger=initialize session request)
If (media=audio+video)
```

**Collaborative behavior.** O+: In a project meeting, if the current topic is the budget, any participant without Manager Role must leave the session.

```
If (TCA.CurrentSession.Type=project meeting) and
  (TCA.CurrentTopic.Type=budget) and
  (TeamParticipantAgent.ParticipantRole!=Manager)
Then TeamParticipantAgent.leaveCurrentSession()
```
**Negotiation strategy.** O+: While negotiating the session schedule, if no full agreement has been reached and more than 75% of the participants agreed, confirm the meeting for the time agreed on

| If   | (VTC.AgreementStatus=NotReached) and                        |
|      | (VTC.numberAgreed=0.75*VTC.numberParticipants) and         |
|      | (VTC.MeetingAttendanceType!=Mandatory)                      |
| Then | VTC.scheduleSession(time,participants)                      |

5 Implementation Consideration

Most key features of the PMS has been implemented and tested for use cases similar to those described in previous section. We have chosen to not use existing rule interpretation engines such as JESS or CLIPS to keep the policy enforcement module as light as possible.

Both policy Management system and V-Team application were developed on top of the FIPA-OS platform. This has the advantage of facilitating the integration between the two systems. In FIPA-OS, agents are developed in Java and the communication is accomplished using Agent Communication Language (ACL). Each agent maintains a profile file written in Resource Description Framework (RDF) that is used for configuration purposes.

The user interface for editing and visualizing policies is implemented using Swing with two important hierarchical views as described in section 3. The PSA is implemented as a daemon on the server, listening for both incoming policy evaluation requests from agents and policy-triggering events.

The policy evaluation request is intended for authorization policies. The request message is composed of (i) the action for which the request is issued, (ii) the agent issuing the request and (iii) the constraints ruling the action. The PSA replies with a message specifying whether the request is accepted or not and under which conditions. Both messages are in ACL. To listen for relevant events, -which actually result from some action-method invocation- the PSA uses the Java Reflection Mechanism. Once an event is identified, the PSA checks the PIB to determine how to handle the event. (i.e. what conditioned actions should take place).

The PIB is, for the time being, composed merely of RDF files that store policies. Actually, RDF allows us to define hierarchies of rules and the groupings of policies while expressing faithfully the internal semantic of an agent’s policy. Policies are attached to the agent. This makes it possible for mobile agent to load and transport their policies while roaming the network. The use of separate RDF policy files is compliant with the FIPA-OS philosophy. Nevertheless we plan to migrate to a database or directory system for central policies and the fact that policies are currently in RDF makes the transition easier.

6 Related Work

The use of policies in management was first introduced by Sloman [1]. His work was the trigger for other research activities focusing on policies. Sloman’s work
introduced policies and showed the power of this concept particularly in the context of distributed systems. However their focus was put on general aspects of policies such as Policy Specification [1][5][6], Conflict Analysis [7], Policy Domains [8] and Hierarchies [9]. Other research groups have focused on the use policies for specific applications. Applications may vary from Network Management as described in [10] [11] to Collaborative Systems as described in [12].

In this work, we have designed the PMS to be partly application-independent. Such a design approach has the advantage of allowing the applications initially designed with no policy abilities to become policy based. That could be achieved in two steps: (i) developing a PSA for this application and (ii) register the application agents with the PSA.

Another important aspect of this work is the use of policy-enabled agents. In previous work [13][14], researchers have used agents mainly to monitor and enforce policies. A new initiative by the FIPA organization [2] attempts to use policies in the context of agent’s platform but remain at a preliminary stage. Furthermore, this specification does not mention any effect of policies on the agent model. In contrast with this, we have considered in our work agents as being the basic entity in the application. An agent wraps every external software or hardware, and each agent manages his own set of policies. The agent refers to the PSA only for application-level and network-level policies. This is interesting in many regards. First we alleviate the burden on centralized entities like PSAs while leveraging the application’s agents. The policies, as a matter of fact, provide the agent with the autonomy and reactivity necessary to his mission and enable the administrator to update the behavior of the agent without altering its code.

7 Conclusion

We have developed a Policy Management System that complete infrastructure for leveraging systems with policy capabilities. It allows the administrator of an application to define, store and enforce policies that monitor the behavior and the interaction between entities of the system.

We have applied successfully the concepts introduced by the PMS to the V-Team collaborative application at various levels of abstraction in such extent that most key features of V-Team are policy-based. The result produced by the prototype we have implemented are thoroughly satisfying in terms of flexibility of V-Team’s agents, their reactivity and the performance of policy enforcement.

Moreover, we believe that policies are likely to become more and more combined with agents especially as an enhancement of agent platforms such as FIPA-OS. We also believe that combining policies and agents by making policies part of the agent’s model is a useful concept in the design of agent-based applications and systems.

Currently, we are refining the policy model to encompass various types of policies and to allow the editing of relationships between policies. We are also designing a distributed version of PSA and considering the issue of context-aware policy management.
References

A Multi-agent Architecture for Intelligent Mobile Agents

Sylvain Goutet¹, Samuel Pierre¹, and Roch Glitho²

¹Mobile Computing and Networking Research Laboratory (LARIM)
Department of Electrical and Computer Engineering
Ecole Polytechnique de Montreal
P.O. Box 6079, Station Centre-ville, Montreal (Quebec) Canada, H3C 3A7
{sylvain.goutet,samuel.pierre}@polymtl.ca

²Ericsson Research Canada
8400 Decarie Blvd., Town of Mount Royal (Quebec) Canada H4P 2N2

Abstract. The client/server technology manages to carry and treat an ever increasing amount of data. However, it is poorly scalable and personalized, and it does not consider the topology of networks. In spite of many weaknesses and the lack of killer applications, multi-agent and mobile agent systems offer more flexibility and reduce network load. They carry their code, where as other applications only send data on the network. This paper proposes a multi-agent architecture which solves this problem by splitting the mobile agent into several cooperating small agents and integrating a notion of neighborhood. Performance measures validated the design of the architecture. Those measures show that the proposed architecture and algorithms improve the intelligence and the use of network resources. As a result, this architecture is suitable for applications where optimising bandwidth is more important than speed, this is the case for many applications in wireless environments.

1 Introduction

Agent Systems for networks applications, from network or dataflow management to Information Retrieval technologies, have gained much interest with the wide adoption of Internet, for the last five years. Even if their utility is still to prove against client-server architectures still performant and evolving, they are due to a bright future once the problems that now limit their capacities find an acceptable solution.

Mobile Agents are at the crossroad of two more ancient concepts: Agent and Mobility. The concept of agent appeared in the field of Artificial Intelligence (AI) in the late 70s and is rather fuzzy, and led to many definitions. An agent is usually defined as a software servant that either relieves the user of routine, burden some tasks such as appointment scheduling and e-mail disposition or sorts the information that is relevant to the user’s current interests and needs [3][5]. This definition has made « agent » a buzzword within both the academic and commercial worlds.
Even if a mobile agent is defined as a special class of agents that has mobility as a secondary characteristic, it is more appropriate to consider mobile agents as the achievement of mobile abstractions, as code, objects or processes. Indeed, mobile agents are mainly studied in telecommunication laboratories and enterprises, with few links to the AI community. They actually use few of the concepts developed in AI, even if much of the current research on mobile agent systems (MAS) focuses on these aspects, but are built on the notions of interpreted programming languages that support mobile code, operating systems independence and object mobility.

This paper proposes a multi-agent architecture allowing to split a mobile agent into several cooperating small agents. Section 2 outlines background and related work. Section 3 presents the proposed architecture. Section 4 analyzes performance results.

2 Background and Related Work

There are three approaches to designing and implementing an MAS [6]. One approach is to use a private language whose features provide the MAS requirements. Compaq™ unsuccessfully explored that approach with the Obliq project [9]. Another approach is to implement MAS requirements as OS extensions [5]. These two approaches were not very successful.

The last approach is to build MAS as specialised application software that runs on top of any OS to provide MA functionalities. The system will actually be composed of two parts: one fixed installed on the servers – the platform – and the agents themselves. Many systems implementing this approach consist of a set of Java class libraries added to the Java Virtual Machine (Aglet, Concordia, Mole, Odyssey, and Voyager). The others are systems using a different, and older, scripting language than Java, with interpreter and runtime support (D’Agent, Ara) [3]. Most of them, faced with the overwhelming popularity of Java, now implement a Java interpreter. Others were even rebuilt completely in Java, like Telescript/Odyssey. All those systems have a server-based architecture and use the sandbox approach for security.

An MAS in Java is built on top of the Java virtual machine (JVM) [8], which provides OS independence and much communication and networking support. It also secures the host machine with the « sandbox » mechanism. This general architecture can be represented by Fig. 1 – ellipses being Java classes.

![Fig. 1. Architecture of a Java MAS](image-url)
The multi-language systems try not to be limited by the restrictions and characteristics of the JVM. Their core is a « Kernel », or « Server » implementing language-independent functionalities like transport, resources allocation, security or thread scheduling. The agents and functionalities implemented as classes/processes or agents will be executed by the appropriate interpreter. Fig. 2 is an illustration of this architecture.

Gray [3] sums well the qualities of mobile (transportable) agents for distributed applications. However, if the size of the agent’s code is too big compared to the amount of data to process, it can affect the performance on network bandwidth. A good balance between the agent capabilities and the complexity of the task it must perform should be observed when using mobile agents. Fig. 3 shows how mobile agents can reduce network load.

Many systems have been developed around the world, but the evaluations described usually remain simple, involving a small number of nodes and good test conditions for the agents. To sum the results, mobile agents do use less network bandwidth [1][2][4], but they still can hardly compete with traditional systems for speed [1] (the test scenario advantages the agent), and even less for load on service machines [4].

Mobile agents have the advantage that they can lead to the development of applications more quickly and that they can reuse all the work done in AI for the past 20 years. They also have several other advantages conferred to them by their mobility,
like ease of customisation, adaptability, or interoperability. Mobile agents systems perform quite well on secure networks, but they need more autonomy and intelligence to react in a more risky or changing environment. Considering their performance, mobile agents are suitable to work specially on these networks. That is the reason why addressing problems such as travelling agent [1], or rerouting in case of network modifications are such important issues. Moreover, it is shown [1] that mobile agents systems get less efficient as interaction with the user increases.

3 Proposed Architecture

Considering the limits of an application based on a single mobile agent (transport of the whole code on every move), we propose to split this agent in many small agents integrated in a multi-agent architecture to enable communication and cooperation. We will describe here the objectives and specifications of this architecture.

The aim of this architecture is to save network bandwidth and other network resources. Mobile agents are supposed to save network bandwidth and to be “network aware”, but in most cases they are not. The common approach even consists in hiding network characteristics behind successive protocol layers and give no way of geographical localisation. We will explore two ways of saving bandwidth: reduce the size of mobile agents and reduce the number of moves necessary to accomplish a given task by improving on search and routing algorithms.

3.1 Agents

Like Esmahi (1999), we will make a distinction between two categories of agents: active and passive or reactive agents. Basically, active agents will act on their own purpose, whereas passive agents will only act upon reception of a message from their environment. Contrary to objects, both are permanent and keep a private internal state which can influence their reaction. This is the difference between object oriented programming and agent oriented programming introduced by Shoham [10]. The proposed architecture must provide a way of looking for other agents or resources. The idea is that an agent arriving on a machine will look for a service rather than an agent. We chose interfaces to represent a service provided by a passive agent, mainly because it helps establish a direct communication between the agents. The corresponding search function, that is not provided by Grasshopper, is implemented in the Registraire agent.

The Registraire is a special agent in our architecture and can be considered as an extension of the MAS. It will keep a trace of all the agents present on the machine and the services or interfaces they offer. An agent arriving on the machine will then subscribe for each interface it implements, and unsubscribe when leaving. The subscription mechanism is controlled by the agents themselves so that they can choose which interface they want to provide. When an needed service is not provided on its machine, the Registraire will search for it on the nearest machines first, to save on network resources. This process is represented by Fig. 4.
Another advantage of this method is to free the mobile agent from the treatment of search and transport errors which can become big. The mobile agent is also less system dependent. The Registraire will also have security tasks, like finding and treating agents that waste system resources. Fig. 5 represents the relations between the different components of the system.

Mobile agents aim at bringing the computing to the data and not the data to the computing [7]. To achieve this, all the needed code is encapsulated in a mobile agent that goes to the data servers. A mobile agent ideally needs on those servers a low level interface, with a great number of fast low level functions, but the servers typically
provide high-level, human oriented interfaces. Such interfaces are very efficient when they meet exactly the user’s needs, but completely useless otherwise [3]. Moreover, the mobile agent must be entirely reloaded when it comes again or for each small modification. The Grasshopper system keeps the agents in cache for reuse, but it
makes any change difficult, and even impossible. The proposed architecture enables reuse of code by encapsulating it in separate agents that will be added permanently to the initial interface of the server, as shown in Figures 6 and 7.

Enabling the reuse of code, we save network resources and automate administration tasks as upgrading of services. Nevertheless, it implies that the system is able to treat many agents and protect the host from malicious or greedy agents, that will try to profit from the system resources without being of any use. The Registraire is able to know all the agents present in the system and keep the necessary information for the calculation of a cost function that will represent the cost of an agent for the system. We can propose a function like:

\[ A/F_{util}+B*D_{util}+C*taille \]

where \( A, B \) and \( C \) are positive normalisation parameters, \( F_{util} \) the frequency of use of the agent, \( D_{util} \) the time since its last use (call from another agent), \( taille \) the size of the memory occupied. Security problems will not be discussed more here. It must be noticed that this architecture does not aim at solving communication problems between many active agents but can be easily extended by tuple space functionalities like JavaSpace or Linda.

3.2 Knowledge

We will consider two types of knowledge: knowledge on the network topology and knowledge on its content (agents, places, and data). The network is represented by a set of places grouped into zones that represent a relation of proximity between the agencies. Basically, we have a set of addresses and zones linked by the relation “is in”. We now have a graph more simple than a representation of physical links on which classic search algorithms can be applied. Complete knowledge of the whole network is not necessary. Fig. 8 gives an example of such a graph.

A resource of the network – place, agent, file, database - will be represented by an address. It will contain the IP address of the machine and the agency where the agent must go to access this resource. It will also contain the name of the agent and the complete path for a file. In order to be as small as possible, a mobile agent must carry
only the necessary information, that is prioritised addresses. To obtain these priorities, the agent must contact other agents that have the needed information. It can use many description languages, but can ask with a simple textual request, similar to those we give to search engines. This leads us to present the information retrieval techniques we used.

4 Performance Evaluation

In order to validate the design of the architecture, we implemented an application using the proposed architecture and made measurements. We focused on the routing algorithms and the use of knowledge on the topology of the network. We will describe this application – the “HuntGroup” – then we will present and analyse the measures.

4.1 The “HuntGroup” Application

The “HuntGroup” application aims at finding a correspondent for a phone call among a group of people. The user only needs to call one number (or one Internet link) instead of many and provides a description of what he is looking for. Then, the agent will do the job. Initially, the application consisted of one mobile agent carrying a static list of correspondents and travelling to each correspondent’s device until it finds the right person. We made the list dynamic so that the agent can be forwarded to any other address at run time. This application can be extended to initiate any call, and even to retrieve any kind of information.

We implemented most of the proposed architecture. The communication KQML has not been implemented but the chosen structure (with interfaces) can be adapted easily. The chore of the application is the “HuntGroup” agent. It is divided into two main classes. The first is the mobile agent itself and contains the communication and transport mechanisms, and one or more itineraries, each itinerary corresponding to one task or subtask. The second class, “AgentItinerary”, represents an itinerary and contains the routing algorithms. The knowledge of the application is kept and treated by the “KnowAgent”. An implementation integrating the knowledge into the HuntGroup agent leads to a mobile agent whose classes are twice as big (33 Ko vs. 16 Ko) and that will have to carry a large amount of data. This consideration justifies the use of the developed architecture and the splitting of the agent into the HuntGroup and the KnowAgent, that will move only when necessary.

4.2 Routing Algorithms

After the development of a simplified version of the application using the Voyager platform, from ObjectSpace, we used for the final version and the measures Grasshopper, from IKV, which provides more agent oriented facilities, and is MASIF compliant. To measure the size of the agents and the network load, we used three Windows NT 4.0 Workstations with an Intel Pentium II 400 processor, and an Ethernet 100 Mbps network. We simulated the itineraries and learning of the agent in
The measures will consider network load as the data to optimize. The execution time stays within a few seconds, which is enough for this kind of application. We will make a comparison between three routing algorithms.

The difference between the three algorithms is the use of the knowledge of the network in the AgentItinerary class. The first version, “simple”, follow the order given by the priorities of each destination without using other knowledge. The “local” version will go first to the destinations located in the zone where the agent is.
A Multi-agent Architecture for Intelligent Mobile Agents

“complex” version will give a priority to every known zone before choosing. Fig. 9 illustrates the difference between the latest algorithms.

We will now simulate the behaviour of the three algorithms in a scenario involving different actors distributed in three towns. Fig. 10 illustrates this scenario. We measured the number of moves for each version for a given succession of requests, differentiating local and regional moves. Figures 11 to 14 show the evolution of the number of moves of each version: «simple», «local» and «complex».

The “simple” version has the worst performances, and the “complex” version does not behave as well as the “local” version. The performances of the three versions get better with learning, as the mobile agents give more and more feedback to the
The bad results of the simple version is due to the fact that the right correspondent is often in the same zone as the user or a machine replying to a similar request. The “simple” version does not use any knowledge on the network and is “lost” quickly. We could expect better performances from the “complex” version. The reason is that the scenario is quite simple, and the “local” version can acquire quickly a knowledge of the whole network and go directly to the right machine, where as the “complex” version is more sensitive to the size of the zones. Moreover, in this application, the agent is looking for only one machine. The “complex” version would
act better to find a group of machines in the same zone, where as the “local” version considers only one machine at a time.

Figures 15 and 16 show the comparison of the three versions for a network with respectively 1 zone and 5 zones. The difference between the “simple” version and others increases with the number of zones. The “complex” version gets better with more zones and at the end of the learning phase, when the application begins to gather knowledge, but does not know “everything”. This observation recommends the “complex” version for dynamics and ever evolving networks where learning is specially important. This aspect is clearer considering regional moves, shown in Fig. 17.

![Fig. 15. Comparison of the three versions for 1 zone](image1)

![Fig. 16. Comparison of the three versions for 5 zones](image2)
In those measures, we limit the tree of zones to one level, where as more levels could bring more intelligence. An other observation is that the proposed architecture brings the needed information closer to the user, since the number of moves and regional moves decreases. When coming back home, the agent visits all the KnowAgents that helped it for this request. Therefore, the number of moves on his way back is the number of KnowAgents it visited plus one. Fig. 17 shows that this number gets close to one. It means that the first KnowAgent visited by the agent has all the needed...
knowledge, thanks to the feedback given by the mobile agents. In a wider network, a 
single KnowAgent would not be able to acquire and hold all this knowledge. The 
knowledge would be distributed between many agents, which makes a search based 
on zones, and the “complex” algorithm more attractive.

Those measures show that we improved the intelligence and the use of network 
resources using the proposed architecture and algorithms. An client/server is much 
less costly for this application (a SIP communication initiation takes 500 bytes) but is 
less customizable and scalable.

5 Conclusion

This paper proposed a multi-agent architecture designed for mobile agents. 
Performance measures validated the design of the architecture. We focused on the 
utility of more complex algorithms that need to carry more data but can be more 
efficient. We found that all algorithms could benefit from feedback learning 
algorithms, and that the algorithms using data on the topology of the network were 
more efficient. Nevertheless, the measures we made could not show an advantage of 
the more complex over a less complex of the latest algorithms. Even if an optimized 
client/server implementation remains the best in terms of performance, the multi-
agent architecture we propose represents an efficient way to cope with a bad or 
inefficient client/server implementation. Performance measures validated the design 
of the architecture. Those measures show that the proposed architecture and 
algorithms improve the intelligence and the use of network resources. As a result, this 
architecture is suitable for applications where optimising bandwidth is more important 
than speed, this is the case for many applications in wireless environments.

References

1. B. Brewington, robert Gray, K. Moizumi, D. Kotz, G. Cybenko, D. Rus, 
   “Mobile Agents in distributed information retrieval”, Matthias Klusch, (ed.), 
2. L. Gannoun, J. Francioli, S. Chachkov, F. Schutz, J. G. Hulaas, and J. Harms, 
   “Domain Name eXchange : A Mobile-Agent-Base Shared registry System”, 
   IEEE Internet Computing, march/april 2000, pp. 59-64.
   science, University of Dartmouth, mai 1995.
5. R. S. Gray, D. Kotz, G. Cybenko, D. Rus. « D'Agents: Security in a multiple-
   language, mobile-agent system », Giovanni Vigna, (ed.), Mobile Agents and 
6. K. Hafner, “Have your agent call my agent”, Newsweek, 75(9), February 
   1995.
7. A. Karmouch and V. A. Pham, “Mobile Software Agents: an Overview”, IEEE 
   Communications, Special Issue on Mobile Agents and Telecommunications, 


A Self-adaptable Agent System for Efficient Information Gathering

Antonio Liotta\textsuperscript{1}, George Pavlou\textsuperscript{1}, Graham Knight\textsuperscript{2}

\textsuperscript{1} CCSR, University of Surrey, Guildford, GU2 7XH, UK
\{A.Liotta, G.Pavlou\}@eim.surrey.ac.uk

\textsuperscript{2} Computer Science Department, University College London, London WC1 6BT, UK
G.Knight@cs.ucl.ac.uk

Abstract. As networks become all-pervasive the importance of efficient information gathering for purposes such as monitoring, fault diagnosis, and performance evaluation can only increase. Extracting information out of large-scale, dynamic networked systems is becoming increasingly difficult. Distributed monitoring systems based on static object technologies such as CORBA and Java-RMI can cope with scalability problems only to a limited extent. They are not well suited to monitoring systems that are both very large and highly dynamic because the monitoring logic, although distributed, is statically pre-determined at design time. The paper presents an active distributed monitoring system based on mobile agents. Agents act as area managers which are not bound to any particular network node and can sense the network, estimate better locations, and migrate in order to pursue location optimality. Simulations demonstrate the capability of this approach to cope with large-scale systems and changing network conditions. The limitations of our approach are also discussed in comparison to more conventional monitoring systems. Keywords. Self-adaptable monitoring; Scalable Information Gathering; Adaptable Information Gathering; Mobile Agents.

1 Introduction

While the size of networked systems grows at an incredible pace, it becomes increasingly difficult to extract information out of those systems. Networked systems and even the networks themselves need constant monitoring and probing for the purposes of management, particularly for fault diagnosis and performance evaluation. For instance, network monitoring entails the collection of traffic information used for a variety of performance management activities such as capacity planning and traffic flow predictions, bottleneck and congestion identification, quality of service monitoring for services based on service level agreements, etc. In this case, a key aspect is that collection of traffic information should be supported in a timely manner, so that reaction to performance problems is possible, and without incurring excessive additional traffic on the managed network. In this article, we highlight the limitations of existing solutions and propose an approach that uses the emerging paradigm of mobile software agents.

Conventionally, information is gathered following a centralized paradigm, where most of the intelligence is concentrated in a single management station which is in
charge of collecting and processing information. This approach has been widely criticized for its limited responsiveness, accuracy and scalability. Typically, the system is partitioned into smaller areas, each of which is monitored by a separate ‘area’ manager. More generally, static decentralized monitoring is realized with an n-level hierarchy of area managers. This is a static approach because the locations of the area managers are computed off-line and do not change after deployment. It scales better than its centralized counterpart but still lacks the adaptability necessary to cope with the frequently changing conditions of large-scale, dynamic networked systems.

To support such adaptation, the information gathering system needs to be able to sense the network state, which is generally dynamic and transient, and react appropriately. Numerous efforts have been devoted to monitoring and probing networks according to a static decentralized approach. Very few, however, have pursued a more dynamic approach to distributed monitoring where the area managers can actually re-locate themselves at run time to adapt to changing conditions in the underlying monitored system. We term this approach active distributed monitoring and present our views on how it can be realized with Mobile Agents (MAs).

Analogous requirements to those assumed herein have been addressed using an approach based on static co-operating agents to build a scalable network measurement infrastructure [1]. The possible advantages of using agent mobility for network management have been discussed extensively in the agent community and are detailed in [2]. Some of these advantages are reduction in network traffic, increased responsiveness, and support for disconnected computing. Furthermore, the authors elaborate on possible applications of MAs to fault management, remote diagnostics, configuration management and performance management.

However, most commonly, in the context of management, MAs are not exploited to the full extent of their capabilities. In fact, the majority of examples presented in the literature use MAs more simply as a mechanism to realize dynamic programmability of remote elements according to the Management by Delegation (MbD) concept, discussed in [3]. We have carried out some work in that direction, identifying key performance issues and studying a possible implementation of MbD based on MAs which we have termed constrained agent mobility [4]. We have then shown the benefits (in terms of added flexibility and dynamic re-programmability) of constrained mobility in the particular context of network performance monitoring by comparing it with conventional approaches based on static distributed object technologies [5] such as CORBA and Java-RMI.

MbD and constrained mobility may be realized with agents bound to single-hop mobility, from manager to remote elements. What is not commonly exploited in management is the agent multiple-hop capability. In the work presented herein we elaborate on the benefits of agent weak mobility [6] –the ability of an agent of carrying code and data when traveling from node to node– for adaptable distributed monitoring. Conversely, agent strong mobility is the ability of carrying also execution state, a property which is not believed to be suited to management applications.

Given our proposal to use MAs as area monitoring stations, a distributed algorithm is required to compute the agent locations both initially and at run time. During the execution of the monitoring task agents will need to sense their environment and take actions in order to adapt to changing conditions and, by doing so, maintain location optimality. Optimality in this case concerns the minimization of the network traffic
incurred by the agent-based monitoring system and of the latency in collecting the necessary information.

A similar problem regarding the optimal placement of $p$ servers in a large network has been studied since the early seventies. This belongs to the class of $p$-center and $p$-median problems, both NP-complete when striving for optimality [7, 8]. Approximate polynomial algorithms, such as the lagrangian algorithm [8], have been proposed but none of them suites the requirement of our agent system. Proposed algorithms are centralized, requiring the network distance matrix at the main monitoring distance. While this is less of a problem in off-line calculations for medium to long-term optimal locations, it becomes an important problem for active distributed solutions in which optimal locations need to be (re)-calculated by the agents themselves. In this case the monitoring station should retain an up-to-date version of whole network topology, which obviously is an unrealistic requirement for large-scale, dynamic networked systems.

In this article we describe our solution to the agent location problem, evaluate its computational characteristics, and demonstrate by computer simulation some of the important features of the proposed agent-based distributed monitoring system. Our algorithm relies on agents learning about the network topology through node routing-table information which is accessed through standard management interfaces. The monitoring system is initially deployed through a “clone and send” process starting at the centralized network-wide station. The same algorithm is also used by the agents to adapt to network changes through migration. Key features of this algorithm are its distributed nature, i.e. each agent carries and runs the algorithm, and its low computational complexity and typical computational time. We discuss the scalability of our approach and its ability to adapt to network congestion and faults.

2 Real-time computation of the agent locations

The agent location problem consists of two phases. Initially, we need to determine what is the appropriate number of agents for a given monitoring problem and compute the location of each of those agents. Subsequently, upon agent deployment the agent system needs to be able to self-regulate in order to adapt to changing conditions. This is achieved by triggering agent migration in a controlled fashion to avoid instability due to continuous agent migration.

In the proposed solution, the location of area managers is neither fixed nor predetermined at design time. Area managers are realized with mobile agents, simple autonomous software entities that, having access to network routing information, can adapt and roam through the network. The distributed monitoring system is deployed by progressively partitioning the network and by populating each partition with monitoring agents. We assume the existence of an agent system supporting weak mobility and agent cloning –i.e. the ability of agents to create and dispatch copies, or ‘clones’, of themselves. Agents are assumed to have access to routing information obtainable from network routers through standard network management interfaces.

2.1 Agent Deployment

The agent deployment process is illustrated Fig. 1.
The number and location of mobile agents is computed by subsequently comparing the monitoring task parameters with routing information extracted from network routers. Starting from the monitoring station, the list of monitored objects (MOs) is matched against next-hop addresses and routing costs to reach those MOs from the current location. This simple matching operation is sufficient for the agent to create a first partitioning of the network. A number of agents equal to the number of partitions is cloned. Each agent is assigned to a different partition and is configured to monitor the subset of the MOs belonging to that partition. Then, each agent autonomously resumes the “partitioning & cloning” process that ends when the number of MOs per partition falls below a given heuristic threshold.

1 Get monitoring task specs (at the monitoring station)
2 Generate 1 MA implementing the task (set MA parameters to the values extracted from task spec)
3 Extract list of MOs from current MA
4 For each MO extract routing info from local router
   * get next-hop node id from current MA location to MO
   * get cost to reach MO from current MA location
5 Estimate cost for current MA to monitor its MOs
6 Use cost to compute number of MAs to be cloned by MA and clone them
7 Decompose task of current MA into a suitable number of sub-tasks
8 For each current MA
   * set its task to one of the above sub-tasks
   * set its list of MOs to a disjoint subset of the total current MOs
   * estimate cost to start monitoring from current location
   * estimate cost to monitor from one neighbor location
   * IF (lowest cost is from current location)
      THEN start MA
   * ELSE {
      migrate to the cheapest location
      GOTO 3
   }

Fig. 1. Proposed agent location algorithm.

The algorithm can be further illustrated by showing the basic steps performed in the case of the simple network depicted in Fig. 2i. Those steps are depicted in Fig 3. Initially, the manager delegates a given task to one agent and starts it at the monitoring station (a). By extracting routing information from the local router and matching them with the list of MOs, this agent estimates the need for an extra agent. An agent is thus cloned, and the original task is decomposed into two sub-tasks, including the redistribution of the MOs between the two agents (b). Then each agent autonomously searches its best location and migrates to it (c). The agent in location 1 is now ready to start since it has estimated that its current location is the one with minimum cost. In contrast, the agent in location 2 decides to share its task with another agent and clones it (d). The decomposition/migration process starts again leading to one agent running in node 2 and the other migrating to node 8 (e). Eventually, the agent in location 8 has found its cheapest location and start executing (f).

It should be noted that the use of cloning results in minimal traffic around the monitoring station. In fact, only two agents leave the station, although the resulting number of agents is three. The cloning algorithm is executed in a distributed fashion (on nodes 1, 2, and 8). Finally, the processing is performed in parallel among nodes at the same level (1 and 2). This algorithm is computed dynamically in the sense that the final agent location depends critically on the network status detected at deployment time.
2.2 Run time Agent Self-regulation

The ability of a monitoring system to adapt to network changes is a very attractive property, especially in view of the dynamic behavior of current and future networks. Network congestion and failures, along with mobile computing result in rapidly changing network logical topologies.

The conventional approach is to achieve adaptability by dynamically changing the routing tree rooted at the monitoring station. This is performed by the routing protocols. Consequently, as a result of congestion or failures, monitoring packets get re-routed through generally longer paths and both traffic and response times tend to deteriorate.
In active monitoring, agents keep sensing the network during their operation and can periodically estimate the cost of alternative locations. Agent migration is triggered when the cost reduction justifies the migration overheads. In our implementation, agents adopt the same logic used during deployment time to sense the network and estimate costs associated to candidate neighbor nodes.

A simple example illustrating agent self-regulation in response to a link failure is depicted in Fig. 2ii. In this case, following a loss of connectivity between node 8 and 13, a new (longer) monitoring path is established between node 8 and node 13. As a result, the central node for the system partition comprising nodes {8, 11, 12, 13, and 14} becomes node 14. Hence, the agent originally located in node 8 will relocate to node 14, bringing the system back to optimality.

This adaptation strategy is based solely on local decisions. An agent knows which nodes belong to its partition and builds cost functions based on the information concerning those nodes, available at the local router. One can argue that the self-reconfiguration mechanism considered as a whole might suffer as a result of this myopic approach. On the other hand, agent myopia has the advantage of simplicity and reduced processing overheads. What the system cannot do is to apply global optimization strategies at run time.

Consequently, the agent system may gradually shift away from optimality if the monitoring task is relatively long and for extreme modifications of the network state. To provide adaptation to those situations we followed the simple approach of re-initiating the whole deployment process. This is more expensive than just migrating a subset of the agents because involves terminating all the agents and starting all over again. Agent re-deployment may be triggered periodically with a period which depends on the system dynamics. Alternatively, it could be triggered automatically by alarms or directly by a human operator.

In practice, our simulations with realistic network topologies (see sections below) has shown that agent re-deployment is not typically necessary because agents tends to end up precisely in the same locations in most cases. Therefore, trade-off design choices between agent migration and periodic re-deployment are necessary for an efficient self-regulating system. In addition, other simple control mechanisms will contribute to the stability of the system. For instance, agents need to incorporate some inertial mechanism to prevent a situation in which minor, high-frequency fluctuations in the network trigger inconsiderate agent migration. Finally, more sophisticated control mechanisms may be considered such as run-time cloning or new agents to respond to rapid increase in system scale. These are not been included in the current prototype because we first tried to approach the location problem in a simple way. Run-time cloning will require mechanisms such as orphan control, containment of agent proliferations etc, which are out of the scope of this paper.

3 Evaluation Methodology

The proposed agent-based monitoring system has been evaluated from different points of view. First, the feasibility of the system depends critically on the agent deployment (or re-deployment) time. We assessed this aspect mathematically to be able to draw
conclusions not only on the asymptotic computational complexity of the deployment algorithm but also on typical deployment times under realistic conditions.

Having proved the feasibility of our algorithm we assesses its goodness through simulations. A crucial point was to run the algorithm for a set of realistic network topologies, composed of routers, links, and hosts. These have been generated using the GT-ITM topology generator [9, 10, 11]. In particular, transit-stub topologies resembling the Internet topology and having 16, 25, 32, 50, 64, 75, and 100 nodes respectively, have been generated in order to assess the sensitivity of the location algorithm to network size. For each network size, simulations have been repeated at least 10 times over randomly generated networks characterized by identical topological features. This was done to guarantee statistical significance of the results. Example 50-node topologies are reported in Fig. 4. You can notice that the actual topologies are significantly different despite other topological features such as average node degree and network diameter are comparable.

![Example 50-node randomly generated network topologies.](image)

In order to simulate IP network and protocol behavior we have adopted the NS-2 simulator from U.C. Berkeley/LBNL [12] and extended it with Mobile Agent capabilities. Agent migration and cloning have been implemented along with the actual agent location algorithm, which is incorporated in each agent. This algorithm has been optimized to minimize the total incurred monitoring traffic. Total hop-distance and maximum weighted distances have been measured for increasing “agents to number of monitored objects” ratios. Those metrics are directly related to the total traffic incurred by the monitoring system and to its response time.

An important parameter we measured was the distance from optimality. To assess how far from optimality our agents ended up we computed the agent locations using three different algorithms: 1) the proposed algorithm; 2) the lagrangian algorithm [8]; and 3) a random location. The lagrangian algorithm is provably near-optimal; hence, by achieving smaller traffic and response time than the ones obtained with it we proved near-optimality of our algorithm. The lagrangian algorithm was computed using the software package SITATION [8]. We also generated the agent location randomly to emulate the worse possible agent distribution.

An important feature of distributed monitoring systems is their ability to scale better than their centralized counterparts. To quantify the potential benefits we have measured traffic and response time for increasing values of polling rate, number of monitored nodes, network diameter, and number of agents. Due to lack of space though we report only the first case.
Finally, we started studying the self-reconfigurability of our agent system by simulating various conditions in which link failures led to increased traffic and response time. We deployed the agent system before the failures; then generated link failures at random locations; assessed the costs associated to agent migration; and finally measured traffic and response time after re-configuration. We repeated the same experiment several times for statistical significance.

4 Adaptation through re-deployment

4.1 Agent Deployment Timescale

The agent location is actually computed during agent deployment. Hence, the algorithmic asymptotic complexity can be estimated by looking at the predominant factors involved from start up until all agents are deployed.

Steps 1-2 of Fig. 1 are performed at the monitoring station, at start up time. Their predominant factor is the cloning time, $CLON_{time}$. In contrast, steps 3-8 may be repeated at subsequent levels of the routing distribution tree (rooted at the monitoring station). They will be repeated at most $R(u)$ times, whereby $R(u)$ is the network radius. Agents running at the same level of the distribution tree, execute independently from each other, in separate physical locations. Hence the computational complexity of the location algorithm can be determined by considering the part that is inherently sequential. Therefore, the complexity is $R(u)$ times the complexity of steps 3-8.

Upon arriving at a node, an agent needs to be de-serialized and instantiated, before executing from step 3. This operation takes a constant time, $DESERIL_{time}$. Steps 3-4 require a number of iterations equal, at most, to the total number of monitored nodes. The dominant cost for each iteration is given by the look-up operation to the routing table to extract the $next\_hop$ and the $cost$ values. Thus, the total contribution of steps 3-4 is $c^*O(N)$, where $c$ accounts for one look-up time. Step 5 involves a number of iterations which, in the worst case, is equal to the maximum node degree, $\delta_{max}$ that in typical networks is significantly smaller than the number of nodes and, typically, does not increase with $N$. The iterations of steps 6-8 are actually performed as part of steps 5 and in the worst case involve the process of cloning and configuring $\delta_{max}$ new agents. Cloning will take a constant time, $CLON_{time}$; the reassignment of the monitored nodes takes a constant time too because it reuses information initially processed during Steps 3-4. Finally, each new agent will require a serialization time, $SERIAL_{time}$ before being sent to its destination. The latter will add a forwarding delay, $FORW_{time}$ and a transmission time, $TRANSM_{time}$.

Therefore the agent deployment time, $DEPL_{time}$ that actually coincides with the time to compute the agent location algorithm, can be expressed as: $DEPL_{time} = \{DESERIL_{time} + c^*O(N) + \delta_{max}*\{CLON_{time} + SERIAL_{time}\} + TRANSM_{time} + FORW_{time}\}*O(R(u)) = c_1^*O(N*R(u)) + c_2^*O(R(u)) \propto O(N*R(u))$.

In practice, $c_1$ is of the order of at most $10E-6$ seconds, since the current technology allows for a number of look-up operations of at least $10E6$ per second. $c_2$ is in the order of seconds since with current mobile agent platforms $[TRANSM_{time} + FORW_{time}]$ is typically in the order of $10E3$ to $10E1$ seconds and $[DESERIL_{time} + CLON_{time} + SERIAL_{time}]$ is in the order of seconds or fraction of seconds [5]. Therefore, if $N \ll 10E6$ then $[c^*O(N)] \ll \{DESERIL_{time} + \delta_{max}*\{CLON_{time} + SERIAL_{time}\} + TRANSM_{time} +
and, consequently, $DEPL_{time} \approx c_2 \ast O(R(u))$. In this case the deployment term will predominate over the computational one and $DEPL_{time}$ will be in the order of seconds times $O(R(u))$.

### 4.2 Distance from Optimality

The distance from optimality of the proposed location algorithm can be evaluated by observing the plots of Fig. 5. The total hop-distance is directly related to the total steady-state monitoring traffic. It can be observed that the proposed location algorithm leads to traffic values that are always smaller than those that would be achieved with the lagrangian algorithm, which is provably near-optimal. Hence, our agent-based algorithm is near-optimal too. In particular, a percentage improvement in the range of 0-3% was measured. It should be stressed once again that the lagrangian algorithm cannot be used to solve the agent location algorithm for the reasons already mentioned in the introduction.

**Fig. 5.** Distance from near-optimality.

It should be noted that, for the sake of completeness, we simulated situations characterized by up to a large number of agents ($p/N=0.4$). However for a more efficient resource utilization, typical “agents to nodes” ratios are envisioned to be much smaller ($p/N=0.1$). The fact that the total hop-distance achieved by placing the agents in a random fashion is very far from our near-optimal solution (38-48% difference for $p/N<0.1$) provides another good justification for the adoption of the agent-based approach. The percentage reduction in traffic with respect to centralized polling ($p/N=0$) is also significant. For instance, for $p/N=0.1$ the reduction in traffic will be greater than 30% and will increase monotonically with $p/N$.

Finally, the fact that the three curves tend to converge for large values of $p/N$ is not unexpected since when $p/N=1$ the number of agents equals the number of nodes. Hence, each of the three location algorithm will equally succeed in placing the agent evenly. The plot which reports the maximum weighted distance (directly related to
(response time) for the three location algorithms is qualitatively analogous to the previous one. However, in this case the agent location curve, though very close to the near-optimal one, does not exhibit any inferior value. In particular, the distance from near-optimality is 0-5% for \( p/N < 0.1 \). This result was expected since the simulated agent location algorithm was optimized to minimize traffic, not response time. Further simulations, not reported here for brevity, proved that near-optimality with respect to response time can be achieved with appropriate alterations to the agent algorithm.

4.3 Scalability

This section evaluates the scalability of the proposed monitoring system from a different viewpoint than the one of Section 4.2. We previously assessed how well the agent deployment algorithm scaled to draw conclusions on its viability. Herein, we evaluate scalability at steady-state by comparing the agent monitoring system with a conventional centralized system. Expectedly, an improvement in performance is achieved with the former approach due to its intrinsic distributed nature. However, the results of our simulations provide a quantitative evaluation.

![Scalability Chart]

Fig. 6. Scalability.

Fig. 6 shows traffic and response time measurements achieved with centralized and distributed polling-based monitoring, respectively. Increasing values of polling rate are required for larger accuracy and timeliness, but incur increasing volume of traffic. In our scenario \( p/N = 0.1 \), whilst network diameter and average node degree are kept constant, hence the linear behavior. It can be observed that the agent solution leads to an approximate 50% reduction in traffic and 28% reduction in response time. Another aspect of scalability is the maximum polling rate that can be sustained by the network. Our simulations showed that the agent solution could sustain polling rates of the order of 200% larger than its centralized counterpart. We then assessed the sensitivity to \( p/N \),
keeping all the other parameter unchanged. This time the curves, not shown for brevity, exhibited a non linear behavior. Both traffic and response time decreased significantly for agent configuration having $0 < p/N < 0.15$. However, for larger values of $p/N$ the improvement was negligible. We concluded that a larger portion of agents is neither convenient nor useful. In fact, the larger is the number of agents, the larger the agent deployment overheads.

5 Adaptation through migration

In this section we present some of our simulation results aimed at evaluating the adaptability of the agent system in face of changing conditions. Agent migration overheads are a major limiting factor but are typically followed by significant improvement in terms of reduced monitoring traffic and responsiveness.

5.1 Migration Overheads

Agent migration overheads are predominated by agent migration time and traffic. In typical general-purpose MA platforms migration time varies in the range between hundreds of milliseconds to seconds [5, 13]. This means that the agent system needs to manage a transient time associated to agent migration in the order of seconds. To improve the persistency of the monitoring system a possible solution would be to implement agent migration through cloning. Instead of migrating, an agent clones another agent and dispatches it to its intended destination. Upon arrival to the target node, the child agent will terminate its parent. This is not a feasible solution for every kind of monitoring task but could often lead to significant improvements.

Another migration overhead is associated with migration traffic. This depends on the agent size which is in turn a function of the complexity of agent logic and of the amount of data transported with the agent. Our agents do not support strong mobility; hence, they do not have to carry the burden of the execution state. In addition, they are designed following the principle of simplicity; then they are relatively small in size.

![Figure 7](image-url)  
**Fig. 7.** Impact of total number of agents on percentage of agent migration occurrences.
An important design choice is the number of agents initially deployed. In fact, the more agents we deploy, the higher the deployment (and re-deployment) overheads will be. A large number of agents also means a larger consumption of computational and memory resources in the hosting nodes. The benefit of a large number of agents is related to a higher level of distribution, followed by generally better steady-state performance of the monitoring system. Another advantage is that, as the number of agent increases, the percentage of agents that need to migrate in face of changing network conditions tends to decrease more than linearly, as demonstrated by our simulation results reported in Fig. 7.

5.2 Migration Benefits

We have simulated a simple scenario in which 2 links located in the vicinity of the central monitoring station fail. Traffic and response time were measured before the failure. After the failure, the routing protocol readjusted the routing tables and full connectivity was achieved. In addition, the agent system reconfigured itself by relocating some of the agents. Steady-state traffic and response time were measured again. Simulations were subsequently repeated for 10 different randomly generated topologies characterized by comparable topological features. Each time a couple of faults was generated randomly and results were averaged.

Fig. 8 shows the snapshot of those two performance indicators (traffic and response time) taken before and after the link failure, respectively. With the centralized polling solution ($p = 0$), both request and response packets get re-routed through longer paths. Consequently, both traffic and response time increase significantly—they almost doubled in our scenario. On the contrary, with the agent system the performance degradation at steady state is in the order of 5-10%.

Fig. 8. Self-adaptation through agent migration.

It should be noticed that, though more extensive simulation will be needed before more generalized conclusions can be drawn, the results achieved so far are very promising. We shall investigate what happens when the number of faults increases to assess the robustness of our system. We have not simulated scenarios in which a fault leads to a
temporary loss of connectivity. Moreover, it will be interesting to conduct more thorough simulations to assess the stability of the agent system.

6 Concluding remarks

In this paper we have presented our progress towards the design of a self-regulating, distributed monitoring system based on mobile agents. While a lot of work has addressed the problem of building scalable, distributed monitoring systems based on the Management by Delegation principles, not much has been done to pursue adaptability in the context of large-scale, dynamic networked systems. We believe that adaptable information gathering is a crucial feature, in view of the pervasiveness of network-centric applications. The interest created by architectures such as SUN’s Jini [14] shows that the scenario in which a relatively large number of simple devices will be accessible across the net is becoming realistic. This introduces a new dimension to networked systems which will become more and more dynamic as we also observe a shift towards all-IP, integrated fixed and mobile network infrastructures.

Of additional relevance to this article is the fact that Jini devices can host mobile code, a feature which would have been unthinkable just a few years ago. However, code mobility represents a serious paradigm shift in the management arena which has not yet found widespread acceptance in the community. It is often claimed that this is due to persistent security and safety concerns which are particularly critical in network and system management.

On the other hand, the benefits of code mobility tend to be undermined by the scarcity of established design methodologies which suit management applications. Code mobility adds a degree of freedom which is hardly conceivable if compared to the well standardized architectures and methodologies refined over the years. The work described herein aims at exploiting this extra degree of freedom. Our initial results are very promising in terms of improved scalability and flexibility achievable with the MA capabilities. We have discussed how agent \textit{weak migration, autonomy, reactivity}, and \textit{cloning} can be employed to design a self-regulating monitoring system targeted to large-scale, highly dynamic networked systems. Another interesting property that might be worth investigating is agent \textit{pro-activeness} to anticipate problems rather than just reacting to them.

Another comment concerns the comparison of the proposed algorithm with approaches based on static distributed object technologies such as CORBA and Java-RMI. If it was possible to accurately estimate the location of the area managers at system design time it would be significantly more efficient to realize area managers with static object technologies rather than MAs. Migration and cloning overheads would be avoided in such case. However, the static approach would not cater for the adaptability offered by the agent solution.

The relatively high costs associated to agent migration supported by general-purpose MA platforms give also an indication of the timescales over which adaptation might be effective. When agent migration times are in the order of a second, the agent system is able to compensate to changes within timescales larger than a second. On the other hand, steady-state performance and scalability will be comparable to those typical of systems based on static object technologies provided that effective methods are adopted to place those objects.
Acknowledgments

The work reported in this paper has formed part of the Software Dependent Systems Work Area of the Core II Research Programme of the Virtual Centre of Excellence in Mobile & Personal Communications, Mobile VCE, www.mobilevce.co.uk, whose funding support, including that of EPSRC, is gratefully acknowledged. More detailed technical reports on this research are available to Industrial Members of Mobile VCE.

References

An Agent-based Approach to Full Interoperability and Allocation Transparency in Distributed File Systems

William Fornaciari§ Vincenzo Piuri‡§ Andrea Prestileo† Vittorio Zaccaria§

§Politecnico di Milano
Dip. di Elettronica e Informazione
Milano, ITALY 20133
{fornacia,piuri,zaccaria}@elet.polimi.it

‡University of Milan
Department of Information Technologies
Crema, ITALY 26013
piuri@dti.unimi.it

†Cluster Reply Milano
Milano, ITALY 20139
a.prestileo@reply.it

Abstract. Modern distributed file system realizations offer only partially resource location transparency, resource location independence, fault tolerance, load balancing, heterogeneity, self-configuration, and simplified user access. Traditional portability techniques developed in these systems become unsuited in highly dynamic environments. To solve these problems within a homogeneous framework we studied and experimented the use of static and mobile agents in a portable environment. In this paper we describe the philosophy, the structure, and the prototype realization of the Agent-based Distributed File System (ADFS). The main properties of this innovative distributed file system are resource location transparency, resource location independence, self-configuration, and heterogeneity of the underlying hardware and operating system architectures.

1 Introduction

The main goal of a distributed operating system is to provide a uniform resource view in a collection of interacting, loosely-coupled computers [1]. The distributed system user must always be able to perceive the same view of the system as well as the same logical and physical resources, independently from his network access point. This is useful, for example, within a Virtual Private Network where users...
may not be aware of the physical position of the resources. In this perspective, a Distributed File System (DFS) plays a fundamental role by creating a unique logical view of the file system resources. The result is a virtual composition named the Distributed Directory Tree (DDT).

A DFS must present various transparency properties to applications. The most important properties are the resource location transparency and the resource location independence [1]. The resource location transparency guarantees that the distributed pathname of a resource does not offer any information about its physical position. The resource location independence ensures the immutability of the resource name even if the physical position is changed.

Traditional mechanisms realizing transparency are usually based on static resource-position binding information, created when a new component unit of the file system is added. For example, to mount the remote file system onto a client directory in NFS [2, 3], the network administrator must create a suitable entry in the mounting configuration file that is used to setup the system tables during bootstrap. If the exported file system location is modified in the network or in the local file-system, information have to be updated manually in every computer of the network to achieve correct distributed system operation.

In the Andrew File System (AFS) [4], the resource-position bindings are partially stored in client’s directories by means of a mapping between the filenames and the identifiers of the atomic portions of the distributed file-system (called volumes). In this case server availability changes can cause incoherence in server-side and client-side mappings that must be manually fixed by the network administrator. The Coda file system [5] improves AFS functionality by adding replication, fault tolerance, and disconnected operation features. Although it represents a substantial step towards solving the server availability problem for opened files, the fully automatic reconfiguration is not yet achieved. This property is useful, for example, when new computers are added to the system.

In the Locus file system [6, 7], a path-traversal mechanism and a globally-replicated mounting table are used by each client to map a resource pathname onto the managing site. Although this globally-replicated mounting table hinders scaling to large and dynamic networks, location and replication transparency goals are significantly reached.

In Sprite [8] distributed resources are accessed via prefix tables [9] stored by clients. The prefix table is in fact only an hint table [10]. When performing a lookup, if the selected hint is not correct the client issues a broadcast query to know which of the servers actually contains the desired resource. The hint table is updated with the results of the query. Even if the Sprite’s adaptation mechanism is a significant step towards dynamic reconfigurability, the large use of broadcast messaging makes this DFS unsuitable for large-scale environments.

The xFS distributed file system [11] realizes the theoretical resource location transparency and independence. These characteristics are achieved by implementing the mapping from a resource name to the storage servers through indirections. Although the design of this distributed file system is very com-
plex, it represents one of the first complete solutions to the problem of location transparency and independence.

In Microsoft Windows NT’s Dfs [12], clients contain a reference to a DFS root server that hosts the upper portion of the DDT. The root server contains a partition knowledge table (PKT) mapping the logical DFS namespace onto a set of servers that physically contain the resources. This client dependence from the root host and the junction points is based on the use of explicit server names to create DDT, leading to low location transparency and independence as well as to reduced operating system heterogeneity. Nowadays networks are becoming highly dynamic environments, presenting challenging problems such as disconnected or weakly connected operations. In these environments the use of static traditional distributed mechanisms is often unsuited.

In our research we experimented the mobile agent technology [13–15] to replace static binding with a straightforward dynamic introspection. Results of our studies and experiments were design and prototype implementation of Agent-based Distributed File System (ADFS). In this system we exploit agent’s ability to explore the network with a minimal set of information to dynamically create mappings between resource-name and position. Besides, ADFS can actively and autonomously assimilate new clients and servers with minimal administration effort when computers are replaced, removed or added. Efforts are confined to the new computers, while in traditional DFS a significant and wide configuration effort is usually required. The global auto-configuration ability with only localized initial configuration can also be exploited as a mechanism to realize hot plug-in (or hot-swap) servers for fault tolerance. On the other hand, since we adopted an highly portable and interpreted mobile code written in Java [16] to realize ADFS, we achieved heterogeneity, interoperability, and portability in a very straightforward way by overlapping the DFS to the local file system.

Due to space limits, this paper focus on configuration and lookup operations only, although a complete prototype of the system has been implemented. This paper is organized as follows. Section 2 describes the basic requirements of our system. Section 3 analyzes the system architecture, while Section 4 presents the implementation and some experimental results. Section 5 concludes the paper envisioning current ADFS research directions.

2 Basic system issues

In ADFS the Distributed Directory Tree (or DDT) is a logical name space, i.e., a set of distributed pathnames, whose structure is virtually overlapped on the physical locations of the distributed system resources (fig. 1).

Pathnames contained in the DDT are directly connected to the resources that they represent. However, to provide resource location transparency, pathnames do not include any information concerning this mapping.

To show how mapping is actually performed, let us introduce some basic concepts. A Distributed Partial Sub-Tree (DPST) is a portion of the DDT composed at least by a root directory. Two DPST are said not overlapping if one
The network directory tree (DDT) is a virtual tree that represents the logical organization of resources available on a network. It is decomposed into distributed directory trees (DDTs) located on different network nodes.

In ADFS (Active Directory File System), the DDT is implemented as a set of non-overlapping distributed directory trees (DPSTs), each of which is resident in a specific computer. This way, each computer knows its own DPST in the DDT, allowing for efficient file lookup operations.

A mobile agent, known as a lookup agent, is automatically created by the client application to perform the file lookup operation. It navigates through the network to find the desired resource, which could be in any computer implementing a DPST. Once the agent finds the resource, it notifies the client application with the resource's network address.

Since the lookup operation is not based on a-priori information contained in the pathname, the ADFS architecture offers resource location transparency. Moving a DPST implementation between computers does not affect its pathnames.

ADFS has been designed, realized, and tested on a hierarchical network model based on a structure containing nodes and sub-networks. Two sub-networks are connected by at least a low-bandwidth link between nodes. The network model supports efficient resource location and independence.

Fig. 1. A virtual DDT mapped on physical locations

Fig. 2. A typical view of a DDT with an empty root and hidden resources.
networks may be either physical or logical. In the first case, the link connects physically one node in each sub-network. In the second case the link consists of complex network path through which the sub-networks can exchange information. In both cases the connected sub-networks are called adjacent.

An agent located in a sub-network determines the next sub-network to reach only on the basis of its knowledge and the adjacent sub-networks. The agent scope is thus dynamic because it varies with the sub-network-relative position of the agent. While logical proximity information is critical to determine the inter-sub-network route, the mobile agent needs more specific information to build its intra-sub-network route. This information is based on the activity status of the nodes in the given sub-network and is updated by ADFS self-configuration system.

3 The system architecture and operation

The distributed system architecture supporting ADFS is composed by an heterogeneous set of computers. Each computer can behave as client, server, or both. ADFS transparently realizes cooperation and interoperability in an innovative way by means of mobile agents.

Each computer of the distributed system contains a computational environment (called location), as shown in fig. 3. The location is composed by a set of system processes and by a set of system modules that provide basic DFS functionalities:
– Lookup Static Agents: processes that manage the lookup requests for a particular application.
– Lookup Mobile Agents: processes that embody the migrating lookup requests made by a particular application.
– Directory Manager Service (DM): used by mobile agents to inspect local resources.
– Look up Table (LUT) Service: used by mobile agents to obtain information about active nodes in the network.
– Agent Manager (AM): used by mobile agents to be accepted in the location.
– Configuration Manager (CM): process that manages the auto-configuration of LUTs.

Let us describe how the system works starting from the lookup system call performed by the application.

The interface of the location towards the application processes (the location API) consists of a set of inter-process calls. To perform a lookup operation, an application calls the lookup procedure. This procedure activates a local lookup static agent associated to that application. This static agent is directed to enhance the performance of the lookup operation. In fact, the static agent is assigned to all process instances of a specific application and furnished of a simple cache of the previous lookup results. In this way, all the users working with that application obtain reduced response time with respect to a common operating system cache.

To realize the lookup operation the static agent creates a lookup mobile agent that inspects all locations of the distributed system until it finds a computer...
holding the given desired resource. Actual lookup mobile agent’s route is hierar-
chically built on top of the logical view of the network. The mobile agent visits
exhaustively all nodes of the sub-network in which it is created. Then, the agent
moves to every adjacent sub-network and repeats the exhaustive exploration of
the nodes within each of them.

Mobile agents perform the navigation by using information about the active
nodes in the local and the nearby sub-networks. A node is active if it is connected
to the network and contains a running location. This information, (i.e., the lo-
cally reachable nodes) is contained in the navigation Look Up Table (LUT) of
the current location. The LUT is simply constituted by a mapping from a sub-
network prefix to the relative set of active nodes. The domain of this mapping,
(i.e., the set of local and adjacent sub-network prefixes) is fundamentally static
and can be locally configured when the computer is added to distributed sys-

tem. Conversely, information about active nodes is dynamic and updated by the
Configuration Manager.

In each location visited by the mobile agent, the local file system resources are
scanned by looking into the Directory Manager (DM) of the location itself. The
DM provides an abstraction layer that transforms the local exported sub-trees
into their respective distributed partial sub-trees. This is done by maintaining,
for each DPST stored in the location, the pair composed by the DPST root and
the local ST root of the exported ST. Interaction between the mobile agents and
DM is very straightforward: by means of inter-process communication, the mo-
bile agent asks the DM about the local existence of a given distributed resource.
The DM checks if at least one of the locally contained DPST roots is a prefix
of the given pathname. In the negative case, the mobile agent does not find the
desired resource locally. Otherwise, the pathname is transformed into the cor-
responding local one by substituting the root of the local ST to the matching
prefix. A local file-system lookup is then executed and the result is returned to
the mobile agent.

When the desired resource is found or the whole distributed system has been
unsuccessfully visited, the mobile agent sends a message with the search result
back to its parent lookup static agent. The static agent provides this information
to the application process that asked for.

With a certain frequency, the mobile agent sends a Check Point Message
(CPM) to the static agent. CPMs are a form of asynchronous information trans-
fer (from the mobile agent to the static agent) about the state of the mobile
agent. The static agent is allowed to generate several agents for a single lookup
when it does not receive CPMs from a given mobile agent within a predefined
maximum time. This is done by using the information of the last received CPM
and the search is restarted from the last point where the dead mobile agent
gave his last vital sign. The robustness of this approach is proportional to the
granularity of check pointing but a very fine granularity could compromise the
time efficiency of the entire system.

Correct location management implies security and networking issues. To such
purposes two additional entities are available in each location: the Agent Man-
ager (AM) and the Configuration Manager (CM). The AM is the agent activator through which mobile agents can request to be accepted and activated in the location. The AM receives the mobile agent’s state and code and verifies the access permissions. If the mobile agent passes the verification, the AM activates it by starting the corresponding thread.

The Configuration Manager (CM) is the process through which the LUT auto-configuration is realized. The CM listens to the network channel for configuration messages sent by other locations and consequently modifies the LUT. Besides when a new computer is activated, its CM broadcasts an activation message to all the active computers. The CMs of these nodes receive the activation message, update their local LUTs, and answer with their activity state. At the end of this automatic configuration process all the active nodes have been configured to reflect the actual state of the network.

4 Implementation and Experimental Results

ADFS was implemented in Java (JDK 1.1.5), on IBM compatible computers running Windows NT. It was also ported on PCs running Windows 95 and Linux. The prototype system was extensively tested in a real geographical network composed by 12 PCs. Computers were distributed on four LANs at Politecnico di Milano namely two in the Milano-Leonardo Campus, one in the Bovisa Campus and one in the Como Campus. Bovisa-Leonardo Campuses were connected by a link at 128Kbit/sec while Leonardo-Como link was of 2Mbit/sec.

Fig. 4 shows the average times of a read operation that include the time to perform a lookup within the network. As can be seen, read operation times of little size files (<10KB) are conditioned mostly by the lookup time, which

![lookup and read times](image)
depends on the path that the lookup agent follows. For files of greater size, the read time is comparable with a read operation performed on a NFS file system since the static agent use a similar direct access mechanism for reading file blocks. For files of growing size the times become linearly dependent on the file size.

Fig. 5 shows the average times of a read operation when the caching of the lookup results is enabled, i.e., the lookup static agent does not always generate a new lookup agent but tries to use the results of the previous lookup operations. In the case that a lookup history for a specific file does not exists or it is not correct a lookup mobile agent is created. This feature exploits the file and directory access locality proper of a typical user behavior and, comparing Fig. 5 with Fig. 4, it reduces greatly the read time of files of little size.

5 Conclusions

In this paper we presented ADFS, an innovative prototype of a distributed file system based on mobile agents. The underlying ideas and the fundamental characteristics were discussed. Resource location transparency and independence through the distributed system architecture have been achieved in a very straightforward way by means of a dynamic introspection based on mobile agents. System adaptivity was obtained easily by self-configuration for very dynamic environments. Portability and heterogeneous interoperability were also provided implicitly by the use of the Java language. Our research is now focused on the issues concerning performance and fault tolerance.
References

A Proposal of PVCs Configuration Methodology Employing Mobile Agents

André Ribeiro CARDOSO
Coordenação de Pós-Graduação em Informática
Universidade Federal da Paraíba, Av. Aprígio Veloso, 882
Campina Grande, Paraíba, Brasil
E-mail: andrec@uece.br

Joaquim CELESTINO Jr
Departamento de Estatística e Computação
Universidade Estadual do Ceará, Av. Paranjana, 1700
Fortaleza, Ceará, Brasil
E-mail: celestino@uece.br

Abstract. This paper presents a research about the technology of mobile agents which has Concordia system as its platform. Concordia was developed by Mitsubishi Eletric Information Technology Center America. In the latest years, the Internet became the main media to access information, data and personal communication. As a result, the overload of information in the bandwidth has become inevitable. Because of the continuous increase of the server connected to the network, the present architecture "client/server" used to connect the computers has become inefficient, therefore relevant changes have been necessary. The development of the technology of mobile agents is considered an alternative solution, in which programs (agents) can move throughout the network to run in different computers. The Concordia's components were written completely in Java, and because of that, Concordia offers the portability and execution of its agents anywhere and at anytime with security, mobility, management and monitoring - all of the necessary characteristics to achieve the perfect environment for the mobile agents. This research was based on the project “A CORBA Distributed Platform with Intelligent Mobile Agents for Service Management (AMI)” of the program “High Speed Metropolitan Network of Fortaleza (REMAV-FOR”). In LARCES/UECE we developed a methodology for PVC configuration using mobile agent, presented in the final part of this work.

1 Introduction

With the growing number of servers connected to the web, the architecture “client/server”, used to connect the computers, has become inefficient and because of that it needs great changes. The use of temporary solutions only transfers the information congestion problem to the Web.

Therefore, it is essential and urgent the development of new technologies to operate and manage the connection among various nodes on the Web. A possible
solution for these problems consists in utilizing mobile agents that assist the user to perform his tasks. These agents can move to the location where data is stored and, with intelligence, select the information that the user needs, saving time, money and bandwidth.

Many programming languages are being developed and implemented for mobile agents framework. Nevertheless there are many problems related to its security that deserve special attention. This new framework needs, consequently, special attention to the mobile agent visits to agents systems, originated from other locations. Before allowing this type of access, for instance, we must guarantee the inaccessibility of certain services of the machine by the visitor agent, avoiding this way the excessive consume of its resources.

This research was based on the project “A CORBA Distributed Platform with Intelligent Mobile Agents for Service Management (AMI)” of the program “High Speed Metropolitan Network of Fortaleza (REMAV-FOR”). In LARCES/UECE we developed a methodology for PVC configuration using mobile agent, presented in the final part of this work.

2 Framework of the AMI Management Service

AMI project main objective is to explore the technology of Mobile Agents and its applications connected to the management of network and services. At first, it proposes a framework based on the technology already in use, like the Concordia that is used here (in this paper), and afterwards the construction of a platform to execute mobile agents developed here at LARCES.

In our project we suggest the following framework divided in five levels:

1. **Application of Services Management Based on Mobile Agents** is composed by a group of intelligent mobile agents that supply the service management. It is made of a group of mobile agents that are dynamically developed and interact in a way to support the activities of service management. Different strategies of the mobile agent behavior and its cooperation can be introduced in this level.

2. **Service of Agent Support** contains a group of services that gives support to the mobile agent execution. Services of name, location and security should exist in this level. It should also contain some basic functions which allow the mobile agent to interact with local services. This layer integrates the specifications MASIF of OMG and adapts itself, when necessary, to give support to the network management and services.

3. **Distributed Support** will provide the functionality to support the interactions and mobility among mobile agents through the middleware CORBA - ORB and JAVA. This allows to isolate the upper layers of the subadjacents technology trough IDL interfaces. In this level, there is also a group of service support such as notification, persistence etc.

4. **Proxy Level** provides the necessary mechanisms to interact with the subjacent agent. It provides the gateway mechanism to interact with SNMP/CMIP or legacy systems management.

5. **Level of Network Management** is made of a group of physical elements and software that are part of the service components and subjacent network. This level is based on the ATM infra-structure at LARCES.
3 Mobile Agents

Mobile Agents introduce a new software and communication architecture by allowing
a program to travel among different computers to run remotely, even among
heterogeneous networks. The idea of remote performance through the transmission of
executable codes among clients and servers has become more and more popular in
recent years in the area of intelligent networks. In the transport of the agent code to
other computers in distributed network, it is not necessary to carry intermediate data
through the network which significantly widens the bandwidth and it can also
avoids the delays of communication.

The task of management delegation can be easily performed by mobile agents that
can be reprogrammed. Mobile Agents can also access the remote resource of a device
for specific management tasks.

The main difference between an intelligent agent and a traditional one is that the
first one not only performs tasks pre-established by the user but also other tasks that
can modify the environment. This characteristic is particularly useful when we are
dealing with management when several times the following situation happens: the
agents are a permanent part of the software that controls the managed entities. The
policy of monitoring and controlling are left to the remote system of network
management.

The concept of mobile agent was originated from three technologies: migration of
processes [1], remote evaluation [2] and mobile objects [3] , all three developed to
improve the Remote Procedure Call (RPC) for the distributed programming.

3.1 Mobile Agents and Management of Network Services

Services and network management is by its own nature a distributed activity that
follows the model “client-server”. In this model a central management entity controls
all the network consisted of managed units. Many of the essential functions in the
network management are performed in the model “client-server” while network entity
with computing ability follows the philosophy proposed by Simple Network
Management Protocol (SNMP) [4] of simple and passive agent structures. However,
this approach has several technical limitations such as scalability, reliability,
performance and difficulties in delegations through network that are becoming larger
and badly distributed.

Network Management using delegations is an obvious alternative for centralized
management. In a network management system there are delegated applications that
work simultaneously as management units as well as managed agents. This delegation
can be controlled and watched remotely. An efficient architecture of distributed
management should address these important issues like reliability, flexibility,
consistence and scalability.

4 Asynchronous Transfer Mode (ATM)

Few technologies have been adopted with such enthusiasm as ATM. In fact, ATM is
emerging as a great and promising network technology due to its velocity, scalability,
flexibility and the guarantee of quality of service (QoS). ATM offers a good
combination of switching packed circuit technique.
The technology ATM uses cells of fixed sizes of 53 bytes. There are the Virtual Path Identifier (VPI) of 8 bytes and the Virtual Channel Identifier (VCI) of 16 bytes. VPI and VCI are the only cells that belong to the same Virtual Connection on a shared transmission medium. ATM operates in a oriented connection model. Before the cells are transmitted from one user to the other, a phase to establish a logical/virtual connection allows the network to reserve the necessary resources, such as bandwidth. There are two kinds of mechanisms to establish a connection: Permanent Virtual Circuit (PVC) and Switch Virtual Circuit (SVC). The first is pre-established at each device along the network and the second one is established under demand, based on procedures of signaling.

A simple final system ATM or a switch does not support all the dimension end-to-end of a VC. Usually a VC is composed of multiple final and intermediate systems, each one supporting virtual links (VLs). Consequently, each final system supports a VC ending and the VLs in its external interfaces, whereas each intermediate system (switch ATM), through where a VC passes, supports multiples VLs in its external interfaces as well as the cross-connections of VLs belonged to the switch. The management end-to-end of a VC is reached through a combination of the management of its individual parts.

The VC is associated with a group of traffic descriptors specifying its characteristics, including the traffic parameters and the class of QoS. VLs inherent characteristics from the traffic of VC of which are part.

### 4.1 Network Management ATM

Two important standardization organizations are involved in standardizing management of ATM network using SNMP protocol for the transport of management information. They are Internet Engineering Task Force (IETF) [5] and the ATM Forum [6]. This paper deals with the first one, ATM management standards of IETF where we will deal with the PVC parameters configuration established among final and intermediary systems of LARCES – UECE.

### 4.2 SNMP for ATM Management

The Internet-standard network management framework, known as SNMP has reached good results in providing interoperable solutions to the problem of network management by enabling effective monitoring and control of heterogeneous devices. Today, SNMP is widely used in network management. Nowadays there are three versions of SNMP management systems: SNMPv1, SNMPv2 and SNMPv3.

Three requirements have to be fulfilled to make an ATM network manageable through SNMP[7]:
- The devices must contain SNMP agents and a collection of management information, named MIB.
- Each device is responsible for the changes in the system behavior, registered in its MIB.
- A manager should be able to exchange SNMP Protocol Data Units (PDUs).

**AtoM MIB**

The differences among the various versions of SNMP have a small effect in relation to the MIBs. The RFC 1695 [8] was developed to specify a MIB for the ATM
network management. This MIB, also known as AtoM MIB, defines the object to manage ATM interfaces, virtual links, cross-connects, and AAL5 entities and connections supported by ATM hosts, ATM switches, and ATM networks. It complies with SNMPv2 SMI, and it is also semantically identical to the peer SNMPv1 definitions. Therefore it can be accessed by both the SNMPv1 and the SNMPv2 management applications.

The primarily purpose of the AtoM MIB is to manage ATM PVCs. Although ATM SVC information is also represented in the management information, full management of switched connections requires additional capabilities that are beyond the scope of the AtoM MIB. Each group of related objects is represented in this MIB as a conceptual table.

5 Concordia

Mitsubishi Electric Information Technology Center America created the Concordia System, with the objective to develop, implement and manage mobile agents applications in order to access information, at any time, place and/or any device supporting Java.

5.1 Concordia Components

Concordia contains multiple components written in Java that together provide a complete framework for mobile agents. Concordia Server is the biggest block in which reside various Concordia managers. Some components have interface and, at any case, each one is responsible for a part of the project in a modular and extensible way. [9]

The components of Concordia are the following:
- **Concordia Server** is the name of the complete component installed and running on a machine in a Concordia Network;
- **The Agent Manager** provides the infrastructure of communication responsible for the transmission of agents;
- **Administrator Manager** provides the remote administration of Concordia;
- **Security Manager** protects the resources and guarantees the safety and integrity of mobile agents and their data;
- **Persistence Manager** maintains the state of mobile agents and objects in transit throughout the network;
- **Queue Manager** is responsible for the scheduling and the guarantee that a mobile agent will be delivered among Concordia Servers;
- **Directory Manager** provides naming service for applications and agents;
- **Event Manager or Inter-Agent Communication Manager** is responsible for registering, transmission and notification of events from one agent to another;
- **Agents Tool Library** is the group of tools and necessary classes that allows the development of Concordia mobile agents.

6 System Prototype

In order to analyze the solution of mobile agents providing the functionality of PVC configuration, a prototype of Concordia was developed, offering a general view of
three phases of this process: configuration, release and reconfiguration of PVC in devices (hosts and switches) of ATM network.

6.1 Assumptions

The system is based on certain assumptions that result in a simpler process of development. These assumptions are necessary to isolate the main issues of this paper, trying to keep the applicability and extensibility of the proposed solution.

1. The functionality of the process is only related to the Configuration of point-to-point PVCs;
2. The class of QoS parameters can be freely configure, however, in order to simplify, the ‘best effort’ bandwidth allocation parameter has to be used;
3. The user has the knowledge of the whole environment (hosts and switches) along the connection path, meaning then that the route is pre-defined and no decision about the routing should be made.

6.2 Implementation Architecture

The architecture of the system is constituted by the components defined below. All mobile agents in the prototype are implemented by using Concordia (Fig. 1).

The Concordia System must be present in each device since it is a mobile agent framework on various platforms. In case the switch does not execute the JVM, the components of the system must reside in another CR that is executed in a separate host responsible for the management of its resources. The Concordia Server provides the necessary intelligence to configure an ATM network. The mobile agents are implemented to execute the different PVC configuration tasks by using the functionality of ATM devices.

The PVC Configuration Manager component, responsible for the management of PVC configuration tasks of the devices, injects mobile agents into the ATM network. It specifies the group of switches along the PVC path, besides initializing the VPIs,

Fig. 1. Implementation Architecture

NC: Network Component
IACMg: Inter-Agent Communication Manager
AMg: Agent Manager
AdmMg: Administration Manager
RAdm API: Remote Admin API
SB API: Service Bridge API

PMg: Persistence Manager
SMg: Security Manager
Q Mg: Queue Manager
DMg: Directory Manager
AT API: Agent Transport API
PVC CM: PVC Configuration Manager
The AdventNet SNMP is a group of classes library written in Java to develop applications and applets for SNMP management networks. We adopted the AdventNet v2c release 3.1 [10] since it supports the JDK 1.1 and higher ones. All other APIs and applications are projected for JDK 1.1, JDK 1.2 and more recent virtual machines. The package can be used to develop applications to manage SNMPv1 and SNMPv2 agents and contact systems of agents using any version of the SNMP protocols at the same time. All Concordia mobile agent interaction with the ATM MIB is done by importing classes of these components.

AtoM MIB contains objects with attributes and values associated to ATM (host and switch), defined according to SMI format. For the prototype used in this paper, the handled objects are the necessary ones for PVCs configuration.

The mapping of our Implementation Architecture in the Framework of the AMI Management Service can be found in [11].

7 PVCs Configuration Methodology

AtoM MIB has as its main focus the management of PVCs and the specification of its establishment, releasing and reconfiguration procedures. The methodology hereby proposed will describe each necessary step for the fulfillment of each phase mentioned above.

An important factor regarding the use of mobile agents in PVC configuration is to provide a uniform way for the ATM network operator to execute this operation. Therefore, it is no longer necessary to have the knowledge of the systems of various devices connected to a heterogeneous ATM network.

The end-to-end VC management using AToM MIB will be illustrated with an example of PVC configuration among the final systems 100.3.1.13, 100.3.1.4 and a intermediary system (switch 8285-100.3.1.2), involved in the project REMAV-FOR that belongs to LARCES-UECE. The VPI/VCI values, ports etc were used in the situation here analyzed [12].
Through the component PVC CM, the user starts the process by entry the PVC configuration data end-to-end, such as connection port and bandwidth of the switches that will be part of the virtual links. This way, a mobile agent is sent to the network to configure the PVC. Initially, with the requirements of the user, the mobile agent executes the task of configuration in the first host, then it migrates to the next switch. After configuring the switch, it travels to the following switch executing its configuration as well. These steps continue until the mobile agent reaches the final host and completes the task of configuration end-to-end. Consequently, it is a sequential procedure, since the mobile agent has to complete each task at each device before moving to the next one in order to complete a PVC. The VPIs/VCIs values are transmitted through the port of the configured device until the port of the next device.

When conditions of recoverable errors occur, the reconfiguration is done through a sequence of negotiations between mobile agents and devices. For example, recoverable failures occur when the VPI/VCI values selected by the PVC CM and already in use or when the bandwidth requested for the virtual link required is not available. When solving these kinds of errors, as well as when facing situations of
negotiation of classes parameters and QoS, the mobile agent may need to return to the last device, by making intelligent decisions. The other kinds of failures can not be recovered. Thus they can not be negotiated.

We shall then present a detail of the steps of a PVC configuration mentioned above.

7.1 Establishment

The PVC establishment process consists of the following phases:

1. *Reserve appropriate VL* – the creation of a VL entry in the VL table (atmVpl/VclTable) by activating the row status atmVpl/VclRowStatus with CreateandWait. The PVC CM initiates to reserve VLs along the route by sending mobile agents to execute SNMP tasks to the ATM devices involved. If no errors occur, a row is created and VPC/VCI values are reserved on that port. The counters of VPCs/VCCs (atmInterfaceVpcs/Vccs) are automatically incremented. The interactions are shown on Table 1.

<table>
<thead>
<tr>
<th>Interaction</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host 100.3.1.1</td>
<td>snmpSet(atmVclRowStatus.11.37.39=CreateAndWait)</td>
</tr>
<tr>
<td>Switch 100.3.1.2</td>
<td>snmpSet(atmVclRowStatus.11.37.39=CreateAndWait)</td>
</tr>
<tr>
<td>Host 100.3.1.14</td>
<td>snmpSet(atmVclRowStatus.3.25.27=CreateAndWait)</td>
</tr>
</tbody>
</table>

Table 1.

2. *Characterize Traffic on the VL* - The virtual link tables characterize the traffic to transmit and receive direction by pointing to the appropriate entries in the atmTrafficDescrParamTable. Multiple virtual links on the table atmVpl/VclTable can point to the same vector in the atmTrafficDescrParamTable.

The mobile agent characterizes the traffic parameters of all Virtual Links associated with the VC through the receive and transmit traffic index in the VL table to the atmTrafficDescrParamTable.

The VLs are activated by setting the row status (atmVclRowStatus) to *Active*. If errors do not occur, the reservation of resources to satisfy the traffic parameters values and the QoS Class for the VL will be completed.

3. *Cross-Connect Virtual Links in the Intermediate Systems associating the VLs to the users application in the final systems* – in the intermediate system (switch 100.3.1.2), the table atmCrossConnecttable should be used to cross the VLs connections. The tables atmVclTable has an identifier column for this purpose (atmVclCrossConnectIdentifier). Different rows in the table atmVclTable that have the same identifier are cross-connected. This is achieved through cross-connect tables.

Before creating a row in the cross-connect table, a unique index must be obtained by using atmVp/VcCrossConnectIndexNext. A get-next will obtain a certain value. The VL cross-connect process consists of the following steps:

1. creating a row in the cross-connect table;
2. obtaining the value of the cross-connect index in the rows of the VL table;
3. activating the row in the cross-connect table;
4. turning on the traffic.
The necessary interactions in the intermediary system are listed in Table 2. At this point the traffic flow must actually be turned on.

<table>
<thead>
<tr>
<th>Destination</th>
<th>Interaction</th>
</tr>
</thead>
</table>
| Switch 100.3.1.2 | 1. Create a row in the atmVccCrossConnectTable
| | snmpSet(atmVccCrossConnectRowStatus 3333.3.25.27.13.37.39=CreateAndWait); |
| | 2. Fill in the cross-connect index value in the corresponding VC table row
| | snmpSet(atmVccCrossConnectIdentifier 13.37.39=3333); |
| | 3. Activate the row in the cross-connect table
| | snmpSet(atmVccCrossConnectRowStatus 3333.3.25.27.13.37.39=Active); |
| | 4. Turn the traffic on
| | snmpSet(atmVccCrossConnectAdminStatus 3333.3.25.27.13.37.39=Up); |

Table 2.

Finally, the traffic in the computers is activated by issuing the values Up to the atmVclAdminStatus row of its tables atmVpl/VclTable.

All the steps above can be shortened by issuing the CreateAndGo value to the row status objects (atmVclRowStatus). This way, it is not possible to obtain a detailed error analyses. Thus the step-by-step process is recommended.

7.2 VL Release

The VL release consists of two phases:

1. Release the cross-connects in the IS – to release the VL, all cross-connects and associated VLs must be released by associating Destroy to the row status in the table atmVc1RowStatus. This will liberate the atmVc1CrossConnectRowStatus value for future use by atmVCCrossConnectIndexNext and the atmVc1CrossConnectIdentifier will be removed from the associated VL.

2. Release the Virtual Links – to restore the associated VLs to the VC, each atmVc1RowStatus entry of the atmVc1Table of each device must be destroyed.

Upon these action, the SNMP agents will release the associated VL resources and decrement atmInterfaceVccs. It is recommended to release the cross-connects before destroying the VLs individually. Otherwise, if the VL is released first, in many implementations, it can be interpreted as a request to change configuration.

3. Release the Traffic Descriptors – to release the traffic parameters associated with transmit and receive directions of the virtual links, the rows of the traffic descriptor table (atmTrafficDescrParamTable) pointed to by the virtual links must be deleted.

7.3 VL Reconfiguration

The main reconfiguration applications consist in the following changes:

1. Traffic and/or QoS Parameter value changes – In this case, an additional capacity of the SNMP agent is not required. The mobile agent takes down the current VC and defines new virtual links with the desired parameter and creates a new VC by following the rules described above.

2. Topology Changes – a topology change, opposed to the reconfiguration described above, requires additional capacity of the SNMP agent, including the hardware/software support.
8 Conclusion

This paper presented a PVC configuration management methodology with the use of mobile agents developed in a Concordia environment. Thus, the PVC configuration manager has an overview of all devices belonged to the network. The user does not have to worry about the system of each switch and is able to delegate the responsibility of configuration to the mobile agent. Although the specific MIB information of each maker is stored in different systems, its access is possible by using the AdventNet. The mobile agent automated the PVC configuration tasks without the need of interventions by the users in the decisions.

The methodology presented is a sequence due to the natural characteristics of the PVC configuration procedure. Like [13] some studies on the mobile agent launching are being done based on parallel methodology since the time spent to configure the nodes tend to be less than if a serial methodology was used.

Although a number of assumptions were made, this paper focus on a great number of relevant aspects to the project and implementation of real architectures of mobile agent systems. The results here obtained are a great indication that operation with mobile agents has significant impact on the performance of management network application.

References

3. http://www.omg.org/about/wicorba.htm “what is CORBA” OMG.
5. IETF WWW Pages URL= http://www.ietf.cnri.reston.va.us/home.html
6. ATM Forum WWW Pages URL= http://www.atmforum.com
10. AdventNet v2c Release 3.1 URL= http://www.adventnet.com
SITA: Protecting Internet Trade Agents from Malicious Hosts

Mark Perry & Qin Zhang

Mark PERRY (Assistant Professor, Department of Computer Science
University of Western Ontario, London, Ontario N6A 5B7
email: mperry@uwo.ca Phone: 519 661 2111 x86644 Fax:519 661-3515
Web: http://www.csd.uwo.ca/markp/) (Corresponding author)
Qin ZHANG (Department of Computer Science, University of Western Ontario,
London, Ontario N6A 5B7)

Abstract: The role of agents and their potential in the electronic marketplace has been discussed widely, but the issue of mobile agent vulnerability to attack, particularly from malicious hosts, needs further development. This paper describes our Secure Internet Trade Agents (SITA) framework that allows for multiple ‘window shopping’ agents to retrieve results, whilst providing anonymity for the user, and providing a manageable key structure.

1 The Secure Internet Trade Agent (SITA) Framework

The SITA model we propose is intended to offer better security for trading with Mobile Agents on the internet, whilst at the same time providing a level of anonymity for purchasers and ‘window shoppers’. It relies on a master agent running on a trusted host that dispatches a series of slave agents to carry out the designated tasks.

One advantage offered by mobile agents is support for concurrent job processing. Thus, a task can be separated into several sub-tasks that can be delegated to several “slave” agents, each of which can execute the task in parallel. We use a layered approach to agent initiation, with one superior agent taking control of the task and dispatching child agents, to give improved security. Since the slave agent is sent to one specific shop server only, the control flow of the code is eliminated (i.e., no comparison is done on that server) and agent itinerary modification is avoided. This allows for confidentiality of the slave agent to be achieved by partial encryption of the agent’s components — namely the agent data. Using this mechanism, an agent protects data that must be used at a particular site by encrypting that data with the site’s public-key. In this way, the data is accessible only when the agent reaches the intended execution environment.

We divide the process of inquiring and purchasing in the electronic marketplace into seven stages — ITA (Internet Trade Agent) initialization, ITA migration, directory search, product information inquiry, negotiation, evaluation, and purchase and delivery. Fig. 1 shows a simple architecture of an electronic market with secure mobile agents that also makes provision for anonymity.
The user creates an ITA and specifies the name of an item and purchase conditions he/she wants to purchase and delegates this task to an ITA.

2) The user sends the instructed agent to an Agent Trade Centre (ATC), which separates the agent from its home address.

3) The trade agent queries a directory agent on the ATC and receives a list of destinations it should visit (for example, ‘ask for all addresses of servers that provide airline tickets’).

4) The trade agent sits on the ATC as a static agent, dispatches one child mobile agent to each destination server using concurrent parallel scheme.

5) The child agent migrates to a market server where it negotiates with the market server and collects offer and reports to the parent agent.

6) The trade agent is responsible for evaluation of the collected offers. It can, after it has finished its task, send a message (by email or mobile phone call or pager) back to its user, giving evaluated result. Alternatively, it queries the database of the ATC for a home address and dispatches a result back to the user with the best offer.

7) The user reviews the offer found by the ITA. If the offer is reasonable, he/she contacts the best-offer server, does the transaction under the terms of the specific signed offer. Eventually, the user receives the purchased goods.

By using this architecture we can make sure that the market servers have no chance of getting any information about the user or about other servers that have serviced such requests. In the case of traditional pseudonyms a trusted third party signs the pseudonyms and thus ensures that in case of need it can identify the user. In our architecture the ATC could do this job, because it is able to identify the user, register the user and ITA together with their home addresses. The ATC can digitally sign the mobile agent and thereby guarantee the trustworthiness of the agents. On the other hand, the agent is ensured and guaranteed by the user who also provides the ATC with her certified personalities (e.g., digital certificates).

For the security of the mobile agents in this system, the ATC can also take the place of the trusted server in one of the trust approaches. The purchasing stage can then be executed over the net between the ATC and the appropriate market server by using secure electronic payment system, such as Secure Electronic Transaction (SET) [1].
2 The framework

When we consider the security requirements in terms of protecting agents from malicious hosts, different needs exist during the different stages of the electronic transaction. In each stage of the above-mentioned activities, a sequence of messages is exchanged between two entities, that is, each stage is a communication session between two parties. In particular, we need security and accountability for each of the sessions.

Throughout the following discussion, we denote that $K$ is a secret key in a symmetric cryptograph session, $K^+$ and $K^-$ are public and private key in an asymmetric cryptograph. $K_A(X)$ is the encryption of a message $X$ using the key $K$ that is generated by principal A. $\text{Sig}_A(X)$ denotes the digital signature of principal A on object $X$. $\text{Cert}(A)$ denotes the digital certificate of principal A. $N_A$ represents a nonce (i.e. randomly generated integer) generated by the principal A. $K_A^+(X)$ denotes encrypting the object $X$ with principal A’s public key, while $K_A^-(X)$ is encrypting the object $X$ with principal A’s private key. Finally, $h(X)$ means to apply a hash function to $X$, create a digest of object $X$.

2.1 ITA (Internet Trade Agent) Initialization

An electronic transaction starts with a user, say Betty (B). Fig. 2 illustrates an initialization by B of an Internet Trade Agent (I) whose unique identifier ($ID_I$) is created by a pseudorandom generator and then B starts the process of requisitioning competitive purchase contracts. The agent I obtains its own public key ($K_I^+$) and private key ($K_I^-$) and is certified by B – thus we have a certificate hierarchy system with the agent I’s certificate $\text{Cert}(I)$ at the lowest level. B authenticates the agent I as her representative by providing her certificate and the identity $ID_B$, denoted as \{\text{Cert}(B), ID_B\}, and specifies her service request. Agent I may learn from B’s previous behavior, guide her and make suggestions. After agent I and B exchange messages interactively, they reach the final shopping requirements (SR). Before agent I migrates, it generates a random secret key $K_I^*$, uses $K_I$ encrypts the SR and current time stamp $T$, denoted as $\{K_I^*(SR, T)\}$. The secret key was encrypted by the public key of ATC, denoted as $K_A^+(K_I^*)$. The whole message carried by the ITC would look like this: $\{\text{Cert}(B), \text{Cert}(I), ID_B, ID_I, \text{Sig}_I(SR, T), K_A^+(K_I^*), K_I(SR, T)\}$

![Fig. 2. ITA initialization](image-url)
2.2 ITA Migration

The instructed ITA migrates to the gateway of the client organization and tries to contact the Agent Trade Centre (ATC). After successful authentication of the ITA’s host and the ATC, ITA migrates to ATC carrying the encrypted shopping requirements. Fig. 3 shows the procedures in this stage. At this point, the user can disconnect from its client computer. The trade agent will continue the request, without bothering the buyer again until the delivery stage.

Fig. 3. Migration to Agent Trade Centre

2.3 Directory Search

Fig. 4 illustrates the hidden home address and directory inquiry. When the ITA arrives at the ATC, the ATC first checks if $\text{Cert}(B)$ and $\text{Cert}(I)$ has been issued by a trusted certification authority. If they are valid and not in the certificate revocation list, the ATC starts to decrypt the encrypted message and check the integrity of the message. The ATC retrieves the secret key by using its private key: $K_A = (K_A + (K_I)) \Rightarrow K_I$, then uses this key $K_I$ to retrieve the shopping requirement $SR$ and time stamp $T$. Also, the ATC checks if the time stamp $T$ is valid. The ATC retrieves the incoming ITA’s public key $(K_I^r)$ from its key management file, computes and compares $K_I^r(\text{Sig}(SR, T))$ and $h(SR, T)$ to see message integrity (if these two values are the same, the

Fig. 4. Directory search
message has not been modified, otherwise the integrity check fails. The ATC refuses the delegation of the user when integrity check fails, otherwise it registers user B if she hasn’t registered before. Only when all answers are certain, the ATC stores the home address of the ITA in a database associated with the identity of user and ITA. The user’s information (e.g., identity and address) is removed from the ITA at the same time. According to the decoded SR, the ITA queries the directory agent in the ATC which keeps information about other web sites and acts as an intermediary broker agent that helps an agent to find business web sites that possess certain required information. The directory agent replies a list of n shop server addresses. Because this communication is executed inside the ATC, and the ATC is a tamper-resistant trust server, we do not use any security technique in this query-response stage.

2.4 Product Information Inquiry

In this stage, the ATC signs the SR and a new time stamp $T$ by using its private key: $\text{Sig}_A(SR, T)$. Then it generates a random secret key $K_d$ and encrypts the SR and $T$: $K_d(SR, T)$, encrypts $K_d$ by using the public key of each destination server: $K_{host}^+(K_d)$. Finally, as shown in Fig. 5, the ITA dispatches one child ITA for each destination server using a concurrent parallel scheme — by employing more than one agent simultaneously in the application. Each child agent carries the certificate of ATC ($\text{Cert}(A)$), the identity of the ATC ($\text{IDA}$), the encrypted message and key, and the digital signature: {$\text{Cert}(A), \text{IDA}, \text{Sig}_A(SR, T), K_{host}^+(K_d), K_A(SR, T)$}. The contacting shop servers in the electronic market only know that they are communicating with the ATC, and have no knowledge of the actual user.

![Fig. 5. Product information inquiry](image-url)
The security of the shop server requires adequate authentication proof, such as an authenticated legitimate and traceable signature of the buyer (the ATC in this case), before accepting further interaction with an agent. Otherwise, the transaction will be denied. A server prevents attacks by denying access to any mobile agent that does not have adequate authentication proof. Additionally, the server of the shop checks the code and data of an agent using anti-virus software before it provides the mobile agent with the required execution environment. Fig. 6 depicts the negotiation stage.

So the shop $S_i$ requires authentication proof from the child agent $C_i$ before accepting its execution request. The child agent $C_i$ gives the shop $S_i$ such proof by showing ATC’s certificate and ATC’s digital signature. While receiving the message $\{\text{Cert}(A), \text{ID}_A, \text{Sig}_A(SR, T), K_{\text{host}}^+(K_A), K_A(SR, T)\}$, the electronic shop $S_i$ first checks the validity of $\text{Cert}(A)$, then decrypts the secret key $K_A$ by using its own private key if the certificate is valid. After retrieving the secret key $K_A$, it decrypts $K_A^+(\text{SR}, T)$ and get the shopping requirements $\text{SR}$ and the time stamp. Thus it can check the validity of the sender’s signature by using the sender’s public key: $K_A^+(\text{Sig}_A(\text{SR}, T)) = h(\text{SR}, \text{ID}_A)$. If the verification process succeeds, the shop $S_i$ provides the child agent $C_i$ with execution environment. The child agent asks for the specific goods under the decrypted SR. The result of the communication is a purchase offer signed and encrypted by the shop $S_i$: $\{\text{Cert}(S_i), \text{ID}_{S_i}, \text{Sig}_{S_i}(\text{Offer}, T_{S_i}), K_{S_i}^+(K_{S_i}), K_{S_i}(\text{Offer}, T_{S_i})\}$.

The child agent receives the encrypted offer (with the valid time limitation $T_{S_i}$), terminates the negotiation, sends back the encrypted offer to parent ITA, and then disposes itself on shop server side.

When a child agent goes to a server that no longer exists, it is able to return back to the ATC and report the failure so that the directory agent can verify and update its database later. If a child agent arrives at one server, whose address has been changed, the child agent is able to send itself to the new address because of its autonomous capability. (We assume that the changed address server retains its origin identity and key pair.)

---

**Fig. 6.** Negotiation with shop server
2.6 Evaluation

The reply messages, which at this point may be less than $n$ because some child agents may have been killed by malicious shop servers, return to the parent agent site, the ATC. The ITA collects all the information, decrypts and verifies shop’s offers. If an offer is not signed or has integrity error, the ITA rejects it and continues the evaluation of those remaining. Eventually, the ITA computes the best offer and queries the ATC database for the user’s host address and dispatches it back to the user with the best offer. For security reason, the best offer $bestOffer$ and time stamp $T_A$ will be signed again by ATC and ITA carries back together with the ATC’s certificate and encrypted report: \{Cert(A), ID_A, Sig_A(bestOffer, T_A), K_B^{-1}(K_i), K_i(bestOffer, T_A)\} Alternatively, the ITA notifies its user about the accomplished task and the final result by sending a email or by mobile phone call or pager.

2.7 Purchase and Delivery

When Betty connects to the Internet, the ITA reports to Betty the best offer it found. Betty will authenticate the ITA first, then review the offer. If Betty thinks the price is reasonable and is willing to purchase, she contacts the shop server that signed the optimal purchase offer with her acceptance. The shop checks its signature on the offer and verifies the valid time period and if everything is as it should be, it cannot repudiate the terms of the accepted offer. Fig. 7 shows the payment procedure.

![Fig. 7. Payment](image)

Betty uses a secure electronic payment system, such as Secure Electronic Transaction (SET), which is accepted by the electronic shop and pays for the requested products. Finally the shop delivers the goods, which may be digital or physical.
3 Security

3.1 Security of ATC against User B and ITA

In SITA, an ITA’s certificate \(\text{Cert}(T)\) is certified by its user B. So it is very important for an ITA to provide its user’s valid certificate \(\text{Cert}(B)\). All the certificates, the ITA’s signature on \((SR, T)\), and the encrypted \((SR, T)\) authenticate the ITA. An unauthorized ITA or user who intends to enter the ATC is refused if they are not able to show an authorized certificate and correct signature at the same time. Furthermore, the encrypted time stamp makes the replaying attack impossible.

3.2 Security of servers against malicious agents

SITA protects servers against malicious agents mainly by authentication and integrity techniques. In SITA, the shop servers in the electronic market require adequate authentication proof before accepting further interaction with an agent. A mobile agent must provide the certificate of the ATC \(\text{Cert}(A)\), signature on shopping requirements, and time stamp created by the ATC to server if it wants to execute on shop server. The shop server will deny execution of a mobile agent that is not authenticated by the trusted third party, the ATC. When mobile agent returns (or send messages) to the ATC having fulfilled its task, the ATC authenticates it by checking shop server’s certificate \(\text{Cert}(S_i)\) and their signatures on shopping offer. The user’s host has to verify returning ITA on ATC’s certificate and ATC’s signature on the evaluated best offer.

If the authentication succeeds, a server may use anti-viral software checks on the code and data of an agent before it provides the agent with the required execution environment.

All servers will follow an effective access control policy and security policy providing the agent with a limited execution environment and grant access rights only for that environment. If an agent looks suspicious for malicious behavior, the server can suspend the execution of the agent forcing it to migrate back, or even destroy the agent. If the malicious agent is sent by shop servers, the ATC records the hostile behavior with the sender’s identity into its revocation list, and will not send a trade agent to that server any longer. If the malicious agent came from the ATC, either the user or the shop server will report to some agent society to verify the reputation of the ATC.

3.3 Security of agents against malicious host

Our SITA framework overcomes most of the malicious host problems. First of all, the proposed ATC is a tamper-resistant trusted third party who provides agent anonymous agent service. The ATC will not engage in any hostile activity against incoming agents. The trust of the ATC is based on using tamper-free trust hardware, administrated by a large commercial institution that has high reputation. It is very unlikely that the ATC turns out to be malicious; if so, not only will the ATC lose its business but also the administrating institution may face legal issues.

In SITA, the ITA uses concurrent parallel mechanism to dispatch one child agent to each shop server in the destination list. Because the ATC is a trusted host, the ITA sitting on ATC as a static agent is resistant to attack by a malicious shop host. More security concerns are related with the protection of the child agents, since they are
mobile. Because a host has to modify an agent in order to give the negotiation result, time stamp, its certificate and digital signature to the agent, a dishonest host may try to alter its state and code or scan the agent for its gathered information. A hostile server may also deny a mobile agent the execution environment, or even kill the agent. If the user sends out only one mobile agent to communicate with several servers in one itinerary, that agent may carry with it sensitive information, which could be used by malicious server for many illegal purposes (inspect other servers’ information, try to revise the information, etc.), by the electronic shop server.

However in the SITA model, the ITA sends out a mobile agent to each server respectively and it returns with the only information given by that particular server. The code of the child agent contains the only information given to that host, instead of control flow statement (such as, ‘do price comparison’) in the code. Thus the code is less likely to be modified. In addition, the child agents do not carry secret keys, offers from other shop servers or other sensitive information (such as a credit card number), since the purchase stage is assigned to the user or the ATC. Therefore, the child agent does not have any information that could tempt the shop server to eavesdrop, intercept or alter. We thus don’t need any detection object carrying with the mobile agent or trace logs on the visited server. We don’t even need to encrypt the child agent as a whole, because the only one thing that a child agent needs to keep secret is the user’s shopping request information. Therefore, we use encryption mechanism to encode shipping request and only the specified server can decoded it. This saves on computation cost and time when compared with a technique that encrypts the whole agent, whilst at the same time eliminating the problem of revealing sensitive information.

A malicious server may deny service to an authenticated child agent with a valid authentication or even terminate the agent. The parent agent can detect hostile behavior against a particular child agent (it knows the identity of the server that each individual child agent visits). The ITA would report the suspicious behavior to the ATC, later the ATC verifies the suspect host and can add the corresponding shop server to a “revocation list” of servers and cease any future transactions with this server if the malicious activity is verified. Moreover, this model can overcome denial of service of some potential hostile shop servers without restarting the whole process. It is unlikely that every shop server that a child agent visits is malicious and will mount denial of service attack on the incoming agent. In particular, we can perhaps assume that in electronic commerce most of the shop servers are set up for doing business and not for the purpose of attacking other agents or servers. Therefore, if only \( m \) out of \( n \) child agents return to the ATC, the ITA can continue the evaluation of the valid \( m \) offers.

### 3.4 Anonymous service of ATC

In the information gathering stage, the ATC replaces the home address of incoming ITA, and signs it using ATC’s private key. The shop servers in the electronic market only know that they are communicating with an ATC. Once provided the offer, an electronic shop cannot refuse to sell the required products under the terms of the purchase offer it issued, because it has previously signed it. The offer can’t be replayed because it has no significance in view of the existence of the time stamp.
During the payment and delivery stage, if the user wants anonymous payment, he can authorize the secure ATC purchasing by providing his/her credit card number (e.g. through Secure Socket Layer [6,9]) to the ATC. Receiving the user’s authorization and credit card number, the ATC will contact the designated shop server, use secure electronic payment system, say SET. The electronic products (such as game and software application) can be transferred to the user through the ATC. Other physical products can be arranged to be delivered to the location of institution that is operating the ATC and then be transferred to the user.

4 Conclusion

SITA is different from the “single agent” approach [4-6], because it applies the inherent ability of mobile agent parallel processing. Instead of using one agent to visit multiple hosts (multi-hops) in one trip, one static parent agent spawns several child agents and dispatches one child to each designated host. Looking superficially, it seems computationally heavy because we run \(n+1\) agents for information gathering instead of one agent. However, these agents are light-weight threads, containing very little code and data. They can be transferred to remote servers very quickly depending on current bandwidth. SITA simplifies security problems, having similar computation cost as one agent and hosts (including visiting shop servers) taken as a whole, or even less. This is because the agent only needs to encrypt the user’s shopping request information, instead of the whole agent. Furthermore, the agent doesn’t need a detection object [7] or the complicated agent structure as suggested by Wang et al in [8], and the visited shop servers do not need to store a Login Data Base nor any other execution trace [9].

Furthermore, SITA has improved on the Kotzanikolaou et al. [10] approach in four ways:

1. SITA provides anonymous window-shopping to users;
2. The Kotzanikolaou et al. approach requires that the user keeps on-line connection for the stage of shop server issuing permission-tokens to parent agent. SITA can operate off-line as long as the user creates and dispatches the parent agent to the ATC.
3. SITA removes the need for agent permission tokens. In Kotzanikolaou et al. approach, such token is mainly to provide the authentication of mobile agents to shop servers. In SITA, the certificates and digital signature of the trusted ATC provide such authentication.
4. Kotzanikolaou et al. approach uses sole public-key cryptography algorithm. Although the public-key encryption doesn’t have key distribution problem, it has the drawback of having higher processing overhead than the secret key. It is about 100-1000 times slower than secret-key encryption. SITA improves on this by using a hybrid encryption scheme (i.e. encrypt message using secret-key cryptography, and then encrypt the simple secret key using public-key cryptography).

In addition, SITA provides a simplified model for improved security in Internet shopping, with the advantage of lessening loads on hosts through the use of the hybrid encryption, small, efficient agents and the utilization of the ATC.
Reference

Applying Mobile Agents to Enable Dynamic, Context-Aware Interactions for Mobile Phone Users

Mark Adler, Jeffrey M. Bradshaw, Michael Mahan, Niranjan Suri

Nokia Research Center
5 Wayside Road, Burlinton, MA USA
{mark.adler, michael.mahan}@nokia.com

Institute for Human & Machine Cognition, University of West Florida
40 S. Alcaniz St., Pensacola, FL USA
{jbradshaw, nsuri}@ai.uwf.edu

Abstract. Mobile agents can play a critical role in enabling dynamic applications on mobile phones. They can carry executable code, making possible effortless downloading of new capabilities and services to mobile phones. When combined with services that support context awareness, user customization, and sensitivity to the mobile phone environment, mobile agents can be used to provide the basis for a rich set of applications. This paper provides an overview of the problems faced in this application domain and outlines the approach we are following in our research.

1 Introduction

Mobile agents are software entities that are capable of moving themselves from one platform or host to another platform or host over the network. Unlike applets, which are pulled in a single hop from server to client, mobile agents determine their own itinerary, which may include a whole series of moves and stops in the performance of its tasks. Since they can carry code as well as data when they move, they can provide their host with new capabilities and behavior in addition to information. By optimizing the location of computing resources, mobile agents support bandwidth-efficient communication, which is particularly relevant given the widening gap between wired and wireless bandwidth. Mobile agents also support disconnected operation—the ability to continue computing on a server while the phone may be unavailable—which is extremely useful in situations with intermittent network connectivity. Finally, mobile agents can be used to move computations to backend servers, thereby reducing the processing requirements and the load on mobile phones (and consequently on batteries).

Mobile agents can be deployed to mobile phones in order to add new executable code to the phone. Once a mobile agent is deployed, the agent can remain on the phone for short or extended periods of time. Therefore, mobile agents can be used to push new services and functionality to phones for either short-term or long-term purposes. Moreover, by temporarily moving agents out of the phone to a backend...
server, services can be swapped in and out of the phone on an as-needed basis—particularly important given the memory constraints typical of such devices.

In this paper, we describe our ongoing research to use mobile agents to enable such capabilities for mobile phone users. We begin by establishing the requirements through a scenario—a researcher attending a conference (such as this one). While this is only one of many possible scenarios, it does serve to illustrate the kinds of capabilities enabled by our mobile agent services. Then, we describe the technical requirements necessary to achieve the scenario. Finally, we describe the current state of our implementation.

2 Motivating Scenario

In the not so distant future, agent technology will transform the way people deal with the logistics of conference attendance. To develop a context for the following discussion of technical requirements and implementation, here is a glimpse of how mobile agents operating with mobile phone devices will enhance the experience of attending conferences of the future.

Registering for a conference, handling travel arrangements, accommodations, and rental cars are tasks that one would like to delegate to a secretary or a personal agent who knows one’s travel-related preferences, calendar constraints, and can handle finding the best air fares, etc. This kind of agent scenario is not novel and does not require mobile agent technology. But mobile agents can be applied to enable dynamic, context-aware interactions that simplify and/or enhance the user experience.

A conference attendee first registers for the conference via the Internet using either conventional means (e.g., using the Web or by voice over the phone) or a hybrid mobile phone–PDA device. After the registration fee is charged to the user’s credit card or authorized by the mobile phone, a mobile agent from the conference is loaded onto the user’s mobile device. This agent enables access to conference hotel information, conference schedules, contact information for conference attendees, conference proceedings, and so forth. The user’s own agent can interact with the conference agent to make hotel reservations, for example, since the user’s preferences are situated with the user’s local agent.

The Mobile Conference Agent (MCA) enables the user’s mobile device in three ways:

• It encapsulates conference and local accommodation information to simplify registration and deal with accommodation details.
• It transforms the mobile phone into an enhanced conference badge, providing a security key for access to conference functions, automating identification for vendors, enabling access to dynamic conference materials on the Web, etc.
• It serves as a gateway to other registered conference attendees to facilitate informal meetings and establish birds-of-a-feather gatherings, find acquaintances among the attendees, and so forth.

This is how mobile agents can simplify daily tasks for a prototypical conference attendee, Pat. After Pat registers for the conference, the conference injects a conference agent into Pat’s mobile phone. The mobile conference agent is able to
communicate with Pat’s personal agent to simplify travel, hotel, and transportation selection. With both agents co-resident on the phone, communication is reduced, and battery drain is reduced. The MCA contains information about conference hotels, flight schedules, the conference timetable, and conference-related discounts. The MCA can relay the travel information back to the conference organizers to help schedule shuttle buses to meet incoming flights. Through the mobile phone, now enhanced by the MCA presence, Pat can browse through the electronic abstracts rather than being burdened by heavy printed volumes only available at the conference. If Pat wishes to print a hardcopy of a particular paper, the MCA will fetch a printable version of the paper and send it to a printer of Pat’s choice.

On arrival at the conference, there is no need to register again and no additional information to obtain; all this has already been handled by the MCA. At the hotel desk, the mobile phone communicates with the hotel agent, verifies Pat’s identity, coordinates Pat’s preferences to find the most suitable room available, and confirms that Pat gets the conference discount rate. Upon checking in, a customized hotel agent is downloaded to Pat’s phone. This hotel agent acts as the room key, allowing Pat to access her room as well as other facilities in the hotel. The MCA confirms that Pat gets the conference discount rate. After Pat is in her room, her personal agent interacts with the hotel agent to reconfigure the room to Pat’s preferences for lighting, TV networks, and radio station for the radio alarm clock. Based on the conference schedule, and Pat’s usual 90-minute morning routine, the alarm can be set to ensure that Pat will make it to the conference on time.

That evening, Pat’s mobile phone filters information from the MCA to find some old friends attending the conference. Using the mobile phone, Pat arranges an informal meeting in the lounge. Agent technology using onboard calendar information and contact information from the MCA is used to get all the friends together. Once together, they decide to continue their discussions over dinner. The MCA provides local restaurant information, each participant’s agent represents their dinner preferences, and the multi-agent planning system finds a suitable restaurant and reserves a table. After dinner, the MCA helps the dinner party discover nearby movie theaters, including what films are showing at what times. A multi-agent planning application can again take the users’ preferences, schedules, and transportation requirements to plan for the after-dinner movie.

At the conference the next day, the MCA enhanced mobile phone serves as an ID badge, and only registered conference attendees are allowed to enter the conference exhibits and lectures. Having the schedule and all the abstracts at hand helps conference attendees choose between parallel tracks. By tracking (in real time) which talks are attended by attendees with common interests, the MCA can influence attendance at talks. Using agent technology, any late registrants can instantly arrange to register electronically, and immediately gain access.

We have outlined just one of many possible scenarios that illustrate how mobile agents could be used to significantly enhance the capabilities of a mobile phone. The capabilities described in this paper can enable applications such as the MCA and several others.
3 Technical Requirements and Implementation

In this section, we outline the envisioned technical requirements and implementation that are necessary for applications such as the MCA.

3.1 Discovery of services and other “relevant” users

In order for agents to discover available services, or for that matter, other agents in the vicinity, there must be a system in place to discover what is available in the vicinity. There are several systems, such as Jini, that provide this facility. Essentially, service providers must advertise their services on the network so that consumers can find them. To avoid being overwhelmed by such advertisements, physical proximity to the services is a good filter.

Service discovery either relies on a strict (and limited) interface, or an open-ended system that relies on a common, but expandable ontology to describe the goods and services. An ontology provides a basis for describing and understanding the services being offered.

Various AI technologies, particularly from the area of Knowledge Representation (KR), will provide a basis for building more powerful discovery mechanisms. In the context of distributed agent systems, the notion of the semantic web [1] is very appropriate. To achieve such a system, the still-emerging knowledge representation formalism DAML (DARPA Agent Markup Language [2]) and its foundation, W3C’s RDF (Resource Description Framework [3][4][5][6], will provide the basis for efforts to populate the Web with content that has formal semantics; thus the semantic web will enable automated agents to reason about Web content and produce an intelligent response to unforeseen situations through matchmaking, management, and control mechanisms based on DAML representations of service descriptions and policies [7].

Sharing vocabularies and models allows automated interoperability; given a base ontology shared by two agents, each agent can extend the base ontology while achieving partial understanding. A base ontology is analogous to OOP systems, where a base class defines “common” functionality.

3.2 Context-awareness

Context-awareness plays an important role in the mobile computing environment [8]. Context acquisition in a mobile computing is provided explicitly by the user or implicitly by monitors [9]. We address two components of context information relevant to our described use case. The first is physical or geographical location. Such data can be provided by GPS when one is outside, or by triangulation based on signal strength in a cellular environment. Indoors, one needs to construct an analogous means of determining location by using beacons and receivers of some variety, for example, the Cricket system under development at MIT [10]. This data can be used to recognize one’s location and orientation, determine the distance between one’s
current location and an advertised service, and even generate a map with directions of how to get from here to there.

The second component, though equally relevant to agent behavior, is the notion of personal context. By personal context, we mean an understanding of the role of the user at any given time. This contextual information should be used to alter the behavior of an agent with respect to the user [11]. For example, during a lecture or a meeting, a phone should switch to silent ringing mode; at a coffee break between sessions, it should switch its volume to maximum to be heard above the commotion. At the agent level, this context is also relevant. Knowing whether the user can be interrupted or not may cause the agent to defer confirmation, and take on a more autonomous role. The agent also needs to be able to coordinate its action with the actions of other people and devices working with the user [12].

3.3 Customization through user preferences

The primary location of the user’s personal assistant is the mobile phone. The personal assistant encapsulates two types of personal data for a user: raw Personal Information Management (PIM) data and preferential information. PIM data is stored and accessed from local and remote PIM servers. Preferential information is directly managed and maintained locally by the personal assistant. The user’s messaging and alert preferences, contact preferences, scheduling preferences, and environmental preferences influence conferencing applications presented by the MCA. Using PIM data, preferences, and the presence of the MCA; the personal assistant will both customize conference services and adapt the physical environment to the user’s liking.

Preferences can be applied bidirectionally. Either the personal assistant or the MCA can initiate the transaction where user preferences are applied. Relative to the MCA, the personal assistant acts as a user interface proxy (conference messages and alerts, etc.) and an interface for service interactions. Relative to the personal assistant, the MCA acts as the interface for all conference services. Three basic patterns for applying user preferences are as follows:

- **Interruption receptivity.** The personal assistant has the means and capacity to evaluate user receptivity to interruption. Knowing the context of the interruption and the user’s current situation, the personal assistant will rate relevance and choose to handle the MCA’s UI-related request in a manner consistent with explicit preferences [11].

- **Service customization.** The personal assistant will customize the services offered by the MCA to conform to PIM data and user preferences. One illustrating scenario details how a user receives conference materials. The user can prefer the receipt of only a subset of the conference papers that pass a keyword filter. Furthermore, the user prefers that these papers are transferred electronically to a user accessible document repository and to then insert “read this paper” tasks for especially relevant papers identified by a context filtering application. Some other scenarios for the personal assistant to customize the
services supplied by the MCA are to tailor lecture and workshop registrations, to choose conference meals, and to schedule meetings with peer researchers.

- **Environmental adaptation.** The personal assistant will also use the MCA agent to adapt to her surroundings. The conference hotel room can be selected upon preferential data: no smoking, near ground level, and outfitted with specific appliances. Dynamic environmental adaptation is also desirable, for example, setting the hotel room morning alarm system (clock radio or wake-up call) based on workout and conference schedule, setting the music, etc. [9].

We intend to explore the representation of preferences in DAML, based on extensions to our DAML-based policy representations and mechanisms [7].

### 3.4 Sensitivity to mobile phone environment

As mobile phones and PDAs converge into a single device with greater communication bandwidth, there are still attributes that separate a mobile phone from a desktop or laptop with a broadband connection. Beyond the physical limitation of the small screen size, mobile handsets have three distinguishing characteristics:

- Limited battery life and, therefore, extreme sensitivity to functions that are power hungry.
- Extremely variable bandwidth depending on location, proximity to cell towers, and type of carrier. Signals may be lost entirely for periods of time, and the greater the distance from the tower, the greater the power requirements.
- Limited on-board computation. These limits include reduced memory sizes, no disk storage, and relatively slow CPU speeds.

So, in a mobile agent world, one can see the utility of off-loading a mobile agent to some external host on the Internet to perform some task in the relative luxury of a richer computational space, and returning later having achieved some computational goal. However, there are reasons why it may also make sense for an external mobile agent to inhabit the mobile phone:

- Privacy and security concerns may more easily be met by performing the computation on the phone to ensure that sensitive personal data is not compromised.
- The mobile agent may transform the behavior of the phone by providing additional functionality.
- The data needed for some computation is local to the phone, and the resulting cost is reduced by running the agent on the phone, rather than transferring the data to and from the network.

This last point suggests that there is some evaluation function that could be computed to determine whether the agent (and data) should be based on the phone or on some network host. We are beginning to define such an equation, and have identified key components. The equation is based on the following parameters:

- **Power consumption.** The amount of power to compute the result. This includes the power to perform the computation locally including the cost of obtaining the data, (the power required to transfer the data to the mobile phone), and the cost of
running the computation locally. Unfortunately, transmitting and receiving data are the most severe consumers of battery power.

- **Capacity.** The ability for the mobile phone to host the application, including sufficient memory and computational cycles. Currently, we are assuming that there is no monetary fee for hosting an agent either on the mobile phone or on some host computer on the Internet.
- **Time.** The estimated elapsed time to reach a result.
- **Risk.** Some measurement of risk of transferring sensitive data into the network.

### 3.5 Security and trust with respect to access to information

The personal assistant is the only exposed application interface to the MCA on the device. The mobile phone is a highly personal, secured device, so the MCA is disallowed direct access to resident applications. The hosting environment on the mobile phone must guarantee this security.

The user has established a trust perimeter about her personal assistant by describing what tasks it undertakes and how it will consider preferential and personal data while executing those tasks. Hence the personal assistant is semi-autonomous — it will need to interact with its sponsor, from simple informative messages to complex queries. For some tasks, the assistant maintains complete autonomy, since the action is within the trust perimeter the user grants the assistant. However, other tasks, say monetary or privacy-related, require user intervention. Granted privileges may be dynamic — the trust a user has in its agent shadows an assistant’s satisfactory or poor performance.

Security and trust are significant issues for mobile agents [13], [14]. With respect to information security and trust, two separate kinds of policies are considered for hosting and interacting with a MCA on the mobile device. The first is to assume that all information the personal assistant gives to the MCA will be public so it is the personal assistant’s direct responsibility to limit the exposure of sensitive personal data or deducible preferences. The second policy is put into force with the MCA upon MCA migration to the mobile device or prior to migration. This policy will bind the MCA or the conference agent system to not expose personal information or preferential constraints. With an enforceable policy in place, the MCA can become more than a proxy for the conference system. It can become a smart delegate for the personal assistant to use and trust. The trust perimeter can be extended to the MCA.

### 3.6 Code mobility

Code mobility is a core requirement for scenarios such as the one outlined earlier. Code mobility allows new capabilities to be downloaded dynamically to mobile phones. In the conference setting, the MCA carries with it new code in order to provide the functionality specific to the conference that the user is attending. We expect that a mobile phone user will experience a variety of situations that will benefit from specialized code being dynamically downloaded to the mobile phone. For
example, each airline might provide a customized flight agent that is sent to a mobile phone when a user makes a reservation. This flight agent could help users make seat selections, meal selections, get information about flights and gates, show maps of airline terminals, and provide access to airline clubs. Similarly, hotels could have customized agents that are downloaded to a user’s mobile phone upon checking in.

Another significant advantage of code mobility is support for small memory capacities. Since code mobility allows code to be downloaded to a mobile phone on demand, code mobility also allows the luxury of removing code that is not currently required. For example, in the previous scenario, before the MCA is downloaded to the mobile phone, other agents left over from previous situations (such as a flight agent from the last flight taken by the user) can be removed. Similarly, once the conference is completed, the MCA could be removed to make room for another agent (such as the flight agent for the user’s return flight back home).

### 3.7 Safe and controlled execution

The mobile phone may become the most personal of devices. It will host or access our private messages, contacts, access keys, and credit and debit electronic payment systems. For this device to host a mobile agent, it must be secured from malicious attack or inept agent behavior. At an agent application level, policy-based mechanisms are a partial solution. At lower abstraction levels, enforcement of these policies through the mobile phone’s base applications, operating system, and drivers must be assured.

Safe execution is particularly important with mobile code. Currently, most mobile code systems rely on code signing to protect an execution environment. However, while code signing provides a means of determining the originator of the code, it is by no means a guarantee regarding the performance of the code. Therefore, even code that has been signed could still be malicious or buggy.

We also feel that as situations become more dynamic and the capabilities of mobile phones improve, mobile phones will see a significant increase in mobile code usage. For example, end users could use mobile agents as active mail by sending executable content to other users. In addition, if multiple hop scenarios arise, agents could also be tampered by malicious execution environments. An example of a multiple hop scenario is a meeting scheduler agent that visits a number of mobile phones to consult user calendars in order to schedule an appointment.

Another requirement is being able to control the resource usage of agents executing on a mobile phone. If phones are to host more than one mobile agent simultaneously, the execution environment must be able to distribute the resources appropriately to the agents. Also, user operations and the environment of the mobile phone must be taken into account. For example, future mobile phone systems are likely to be packet-based, allowing more than one communication channel to be active simultaneously. While a user is communicating over a voice channel, agents might still be allowed to communicate with other agents or services. Moreover, the total bandwidth available to the mobile phone may vary over time, as the number of customers in a cell change or as the user moves across communication cells. Under such circumstances, the
execution environment must be able to limit and distribute the bandwidth used by agents so as to not interfere with the user’s voice communication.

### 3.8 Implementation architecture

The following mobile reference architecture can be used to realize the conference attendance scenario or another similar application that requires mobile agents and mobile devices [15]. The reference architecture identifies the required, high-level components for agents to be discovered, moved onto the mobile device, interact with a user’s agent on this device, and interact with remote services. These mobile agents can be self-contained, or can act as a networked federation to provide a more dynamic service to the mobile phone user vis-à-vis the user’s personal assistant agent.

The personal assistant (PA) lives and runs on the mobile device. The 2.5/3G network and the low-power wireless network will provide coarse and fine-grained information to ascertain device location. In addition to interacting with mobile application services, the personal assistant will communicate with PIM services, location independent services, and location dependent services. One special type of location dependent service, a local ad hoc service, can be made available across the low-power wireless network. Mobile application services are unique in that they can dispatch a mobile application agent (MAA) to a host phone. If a desired mobile application service is detected from across the low power network, a MAA can migrate to phone either across that network or the 2.5/3G network, dependent on agent size and bandwidth considerations.

Once on the mobile phone, the MAA will be hosted inside the safe execution environment. All user interactions and phone application services requested by the MAA are routed through the PA. All other resource requests are supplied or denied by the safe execution environment.

The safe execution environment will be based on the NOMADS mobile agent system [16] and the capabilities of the KAoS agent framework [7]. NOMADS provides unique capabilities for strong and forced mobility and safe execution of mobile agents. Strong mobility allows the execution state of an agent to be captured and moved with the agent from one host to another. In addition, NOMADS relies on its state-capture mechanism to support forced mobility, which allows the system to move agents from one host to another at the discretion of a user or agent management facility, potentially in a completely transparent manner to the agent. Such forced mobility is essential for applications involving load balancing, process migration, devices shutting down, and so forth. Safe execution of agents is based on the ability of NOMADS to control the resources accessed and consumed by agents. The resource control mechanism allows control over the rate and quantity of resources used by agents. Dynamically adjustable limits can be placed on several parameters including the disk, network, and CPU. These resource control mechanisms complement Java’s access control mechanisms and help in making the NOMADS system secure against malicious and buggy agents. NOMADS derives its unique capabilities from a custom Java Virtual Machine called Aroma.
Complementing the NOMADS features, the KAoS agent framework provides mechanisms for overall management of agents grouped into domains. The KAoS domain manager serves as a policy decision point to determine whether agents can join a domain and for policy conflict resolution. Guards interpret policies that the domain manager has approved and enforce them with appropriate native mechanisms. The domain manager ensures policy consistency at all levels of a domain hierarchy, notifies guards in the event of a policy change, and stores policies in a secure repository. These policies are stored in an implementation-neutral format, currently very simple but soon to be based on our DAML policy representation. Because the library expresses the policies declaratively, authorized entities can analyze and verify them in advance and offline, maximizing the efficiency of execution mechanisms.

KAoS policy-based agent management includes features supporting authorization, encryption, and access control while adding the ability to represent policy for NOMADS resource control mechanisms. But because of our focus on agent systems, KAoS goes beyond these typical security concerns in significant ways. For example,
the KAoS architecture introduced the concept of agent conversation policies. The agent-to-agent communication process uses appropriate semantics to form, maintain, and disband teams of human and software agents assisting the user with a given task [12]. In addition to conversation policies, we are developing representations and enforcement mechanisms for mobility policies, privacy policies, domain registration policies, and various forms of obligation policies.

References

Using Mobile Agents for Managing Control and Alarms in Urban Infrastructures

Alejandro Quintero, Dougoukolo Konaré, Samuel Pierre

Mobile Computing and Networking Research Laboratory (LARIM)
Department of Electrical and Computer Engineering
École Polytechnique de Montréal
C.P. 6079, succ. Centre-Ville, Montréal, Québec, Canada, H3C 3A7
E-mail : Samuel.Pierre@polymtl.ca

Abstract. This paper proposes an application of mobile agents for managing control and alarms in an integrated system dedicated to coordinated management of urban infrastructures (SIGEC). This system allows an ordered planning of the required work in an urban sector as well as an impact and the cost reduction of the interventions on the urban infrastructures. The SIGEC is based on a cooperative system which integrates a set of operating system (SIDEX), each of them being associated with a specific urban system (Sewerage, Waterworks, etc.). Dedicated to the management, regulation and interactive and dynamic monitoring of urban infrastructures in an efficient and correct way, the main objective of this system is to integrate the set of SIDEX into a single coherent environment that can help different classes of user achieve their tasks, their roles and their responsibilities within the municipal administration. In this context, the information can be presented in different forms: video, pictures, data and alarms. One of SIGEC’s objectives is the real-time management of urban infrastructures’ control mechanisms. To carry out this process, the alarm control agent creates a mobile agent associated with the alarm, which is sent to a mobile station and warns an operator. SIGEC is provided by different measurement and monitoring instruments installed on some system’s elements to be supervised. Preliminary implementation results show that SIGEC supports effectively and efficiently the decision making process related to managing urban infrastructures.

1 Introduction

The rehabilitation of urban infrastructure is currently one of the main concerns of North America’s municipalities. Urban system management has always been the result of the collaboration of various actors. However, currently, these urban infrastructures have more constraints due to the transfer of responsibilities as well as the decrease of human and economic resources. Nowadays, only a concerted effort among the groups that participate in urban infrastructure management and continued monitoring will allow the rehabilitation of the infrastructure. It is in this context that the LARIM research lab has started work in an innovative project called SIGEC and
aiming to develop an integrated system for the coordinated management of urban infrastructure. This system should ensure the optimal functioning of some urban infrastructures of a municipality.

Currently, at the planning level, the maintenance level or the rehabilitation level, decisions are frequently made without consulting all involved actors. The integration of automated tools in everyday activities is not common. Furthermore, the developers of urban system management applications have not adopted the concept of integration. These applications are generally proprietary applications making its adaptation as well as data exportation quite costly. Likewise, much of the data cannot be transferred to other applications, thus making their use quite limited for the decision making process. Therefore, reuse and integration of these tools are very difficult.

On the other hand, since the inception of the theory of Multi-agent systems (MAS) there has been an interest in studying and modeling the behavior of various agents that cooperate to solve a problem or to carry out a specific task. For example, in [1] multi-agent and knowledge-based systems have been used to design an Electronic Market Place, or in [4] where the multi-agent systems have been used to design and support a call center.

Mobile agents have been a research topic of interest for several years, yet this research has for the most part remained within laboratories and has not experienced a wide-scale adoption by industry. The development of the WWW application, however, has dramatically stimulated interest in this area of research by offering the possibility of a widely deployed application that could use mobile agent technology. Mobile agents are a particular type of software agent, having the capability to move from one host to another. A software agent can be defined as [2].

"... a software entity which functions continuously and autonomously in a particular environment ... able to carry out activities in a flexible and intelligent manner that is responsive to changes in the environment ... Ideally, an agent that functions continuously ... would be able to learn from its experience. In addition, we expect an agent that inhabits an environment with other agents and processes to be able to communicate and cooperate with them, and perhaps move from place to place in doing so."

A number of advantages of using mobile code and mobile agent paradigms have been proposed [14][15]. These advantages include: overcoming network latency, reducing network load, executing asynchronously and autonomously, adapting dynamically, operating in heterogeneous environments, and having robust and fault-tolerant behavior.

This paper explains how MAS and mobile agents can be used to model systems that describe cooperative environments as for example the SIGEC. The SIGEC architecture has been designed using elements of agent technology.

The application area of the management of urban infrastructures (MIU) has an interesting combination of characteristics: processes in MIU take place in a distributed manner; the requirements for system supporting an MIU are a dynamic nature; the domains are knowledge-intensive; and the supporting systems should be easy to maintain. For example, these characteristics are combined in a transparent manner for designing and specifying interacting reasoning components in the SIGEC system and in the other systems, like in DESIRE [5].
More details on SIGEC architecture are provided in Section 2. Section 3 describes the agent structure. Section 4 presents the alarm management system. Section 5 presents the specification of the system, and finally Section 6 presents the conclusion.

2 SIGEC Architecture

The integrated system for coordinated management of urban infrastructures must ensure an optimal functioning of urban systems. Each of the urban systems considered is managed by a specific SIDEX (integrated operation system), dedicated to the management, supervising, and dynamic monitoring of a specific urban infrastructure in an efficient and correct manner. The objective is to integrate the set of SIDEX in a single coherent system for the SIGEC’s users, according to their tasks, roles, and responsibilities within the municipal administration. As part of this objective, an intelligent system is developed. This system allows an ordered planning of the required work in a single urban sector, thus reducing its impact and the costs of the interventions on the urban infrastructure.

Each of the urban systems considered is managed by a specific SIDEX. Some of the urban systems considered and managed by SIGEC are: Sewerage System, Waterworks System, Public Lighting System, Road System, and so on.

SIGEC allows the coordination of the set of urban systems, as shown in Fig. 1. It is important to note that information can be presented in various forms: video, pictures, data, and alarms. The information is managed by different measurements and monitoring instruments installed on systems that must be supervised.

Because of the integration principle, SIGEC is like an orchestra conductor that manipulates the information regarding the different urban infrastructures under the responsibility of the municipality. SIGEC allows the users to know and react to the current state of the urban system as well as to the future state based on projections and

Fig. 1. SIGEC Architecture
extrapolation of current data.

The use of SIGEC for the effective urban infrastructure management is the fruit of the collaboration of various SIDEX and their individual contributions, that allow a more global view of the activities related to the planning, programming, operation, execution and supervision of jobs. Moreover, it may also involve the collaboration of other tools (i.e., GIS) that work on similar topics or on topics of general interest. The advantages of cooperative work are the following: the feasibility of combining various sources of information and knowledge, the ease of detecting and correcting errors, and the improvement of the quality of knowledge.

The SIGEC can be seen as being composed of four elements:

- A group of agents: IDSS agents, communication agents, and so on. Each agent has some characteristics [18]: autonomy (all agents are in full control of their own processes), social ability (all agents are able to communicate and cooperate with other agents), pro-activeness and reactiveness.
- A group of mobile agents. These are useful for applications that need to respond in real time to changes in their environment, like alarm management in urban infrastructures, because they can dispatched from a central controller to carry out operations directly at the remote point of interest.
- A set of tasks to be carried out.
- A set of resources: the urban infrastructures and all information associated with them.

2.1 SIDEX Architecture

Strictly speaking, the generic SIDEX is an integrated system dedicated to the operation of a generic urban infrastructure. All structures that have been studied have common features that merit that they be grouped into a representative system, which is a generic system.

Urban systems have characteristics proper to each, which allow them to be

![Fig. 2. The knowledge in the SIGEC](image-url)
distinguished from other systems. Not only the physical (i.e., dimensions), structural (i.e., composition), environmental, and economic (i.e., cost) aspects must be taken into account, but also those aspects that are pertinent to the relation between the system and its actors (users).

IDSS is an intelligent system to aid in decision support. It is made up of a knowledge-based system and inference mechanisms. The knowledge-based system stores experts' knowledge as well as solutions to past problems (Fig. 2). The inference mechanism is the one that guides users to making the correct decisions. IDSS is composed of two different levels:

- **Global Level (G-IDSS):** At this level, the knowledge stored comes from experts in each SIDEX. At this stage, IDSS helps users in decisions related to design, planning and global task coordination in which various SIDEX participated.
- **Local Level (L-IDSS):** At this level, only knowledge that is related to the specific SIDEX is stored. This IDSS aids in operative and maintenance decisions that relate to the urban network, represented by the SIDEX. Each SIDEX integrates an IDSS, which manages and stores specific knowledge of each urban infrastructure system. The way the information is structured as well as the decision aiding systems are generic, though the knowledge stores as well as the problems that are dealt with are specific to each SIDEX. The IDSS is an analysis and aggregation system enables the SIDEX to make strategic choices in terms of technical interventions on municipal infrastructures. These interventions belong to specific infrastructure (Sewerage, Waterworks, and so on).

### 3 Agent Structure

In our system, the model agents are homogeneous as to their architecture (their inner logical structure), their operation (internal dynamics), control specification (goal, plans and strategies specification), functionality (what they can do), global goals, ontology (Ontology, as Gruber [11] describes it "explicit specification of a conceptualization", provides a vocabulary for talking about a domain) and used communication (ways and forms of communication).

The homogeneity of the ontology is very important, because with this there is no need for a translator or for an interpreter of knowledge and concepts between the model's agents, even if their knowledge representation differs. In [12] we can find the most essential criteria used during the design of SIGEC ontology. These criteria are: clarity, coherence, extendibility, minimal encoding bias and minimal ontological commitment. In our model exist different ontologies: the alarm ontology, the complaint ontology, the location ontology, and the intervention ontology.

In the *functionality*, all the operations or tasks on the system that are known by the agent or other agents require that are described.

The *control* of an agent is made up of the specification of the goals, the intentions, the plans, and the strategies. To reach these goals, the agents require cooperative working.

The *Knowledge Communication System*: Different SIDEX and IDSS manage diverse types of knowledge and require this knowledge to circulate among them,
mechanisms for sending information to others, the capability of every SIDEX to communicate simulation results, task results and query results to form knowledge as a result of their simulation. At the SIGEC stage, there are different levels at which communication takes place: communication among SIDEX to collaborate and carry out task execution; communication between the SIDEX and the global control and planning system to carry out the coordination of a set of tasks on the urban infrastructures; communication between the system’s users and the SIGEC, and communication between control and monitoring elements, the GIS and SIGEC. In SIGEC, an homogeneous ontology is used to allow the communication among all its elements.

Mobile agents have many characteristics that enable them to enhance managing control and alarms in urban infrastructures. Mobility is obviously one of the most important capabilities, and we can certainly benefit from it. However, other agent capabilities also lend themselves for coordinated management of urban infrastructures. Mobile agents are by nature autonomous, collaborative, self-organizing and mobile. These features are not found in traditional distributed programs, and enables SIGEC to implement completely new approaches for managing alarms. Some advantages of mobile agents for managing alarms in urban infrastructures are:

- **Software deployment**: It is an evolving collection of interrelated processes such as release, install, adapt, reconfigure, update, activate, deactivate, remove and retire. The connectivity of large networks, such as Internet, is affecting how deployment is performed [13]. In order to support software deployment, the agents-based technology must: operate on a variety of platforms and networks environment, ranging from single sites to the entire Internet; provide a semantic model for describing a wide range of software systems in order to facilitate some level of software deployment process automation; provide a semantic model of target sites for deployment in order to describe the context in which deployment process occur and provide decentralized control for both software producers and consumers.

- **Overcoming Network Latency**: It will always be faster to send a message to a network node to execute predetermined, resident code, rather than send a mobile agent to the node. However, such an architecture requires that all response and reconfiguration actions be predefined, replicated and distributed throughout the network. The response mechanism then constitutes, in effect, a large distributed database, raising serious administration problems concerning configuration management, consistency and transaction control. Innovative responses must be transmitted at least once to each affected node, either by conventional network means, a series of messages, or by a mobile agent. Of these choices, the mobile agent technique offers the fastest response.

- **Reducing Network Load**: One of the most pressing problems facing current alarms management in urban infrastructures is the processing of the enormous amounts of data generated by the measurement and monitoring instruments installed on some system’s elements to be supervised. Mobile agents typically process most of this data locally.
• **Adapt dynamically**: Agents can perceive their environment and act on their own to solve a problem.
• **Robust and fault-tolerant**: Mobile agents’ ability to migrate between hosts makes them attractive for implementing fault-tolerant systems.

### 3.1 Intelligent agents

- **Organization agent (OA)**: This agent represents the different organisms that relate to the urban system. There is an interrelation among the different organizations.
- **Complaint Agent (CA)**: This agent is in charge of dealing with complains or claims in a general manner.
- **Geographic Identification Agent (GIA)**: This agent supplies the geographic location on any element in the urban infrastructure.
- **Urban System Agent (USA)**: It is the agent in charge of managing the urban system’s inventories and its state.
- **Measuring Agent**: This agent is in charge of manipulating and analyzing all information related to measuring, control and monitoring devices. Additionally, it’s the agent that is in charge of interacting with the alarm mobile agent that will go to the operators or technician’s machines when a specific task is to be executed.
- **Intervention (Task) Agent**: This agent is in charge of managing all tasks related to a specific SIDEX’ maintenance, rehabilitation and construction. This agent interacts closely with the IDSS agent to develop in optimal manner rehabilitation and maintenance plans.
- **IDSS Agent**: This agent is in charge of supporting the decision making process and manages the knowledge-based system. At the SIDEX level, the knowledge stored is directly related to the corresponding urban infrastructure and aids in local decision making, usually associated with maintenance and operational issues.

### 3.2 Mobile Agents

- **Alarm Control Agent (ACA)**: One of SIGEC’s objectives is real-time management of urban infrastructures’ control mechanisms. To carry out this process, the alarm control agent creates a mobile agent, associated with the alarm, which is sent to a mobile station and warns an operator. This agent needs the geographic location on any element in the urban infrastructure. Normally, this type of information is supplied by a GIA. Thus, the agent is in charge of interacting with the GIS and communicates with it to ask for the necessary information to carry out a given task. This agent interacts through a simple interface with the operator, and if the operator needs information related to the problem, the agent goes to the SIDEX and interacts with some of its agents. After its creation, the new agent participates fully in the running multi-agent system. It has a permanent interaction with the history of each element.
and the IDSS, thus allowing the IDSS offer more precise help when required. This agent takes each complaint or claim and analyzes from the point of view of interventions that are carried out and the behavior of some of the infrastructure’s elements. In this context, the agent creation's concept is different to the agent creation in the system with deliberation SWD [3]. In SDW, the agents are capable of deliberation about the creation of new agents, and create a new agent, on the basis of this deliberation.

- **Control agent**: This agent is in charge of coordinating the tasks that are to be executed and the agents to do them. This coordinator must decide if it is necessary to change the structure of the tasks and the processes to carry them out, or just a part of it. The decisions are made based on the knowledge and experience of the agent, everything is stored in its knowledge-based system.

## 4 Alarm Management System

The Alarm Module Management (AMM) is an important aspect of real-time management of urban systems. Its main objective is to ensure management of any alarm generated by an electronic equipment within a SIDEX. Those equipments are controllers, detectors, cameras, that send data and real-time video to a specific SIDEX. When data comes into each SIDEX, they are processed automatically before storing them into the database for future analysis on an urban system by specialized software. When the data sent by an electronic equipment to a SIDEX is below a certain value, an alarm is generated and the AMM has to take care of it. While data is processed at the SIDEX level, alarms are processed at the SIGEC level by the AMM, because alarms are to be coordinated before taking action. Actions taken by the AMM are to contact people, to publish information and to ensure the follow up of the alarm using the mobile agent technology. This section presents the requirements and analysis needed to design an AMM. First of all, an alarm has two static states:
• alarm has been treated and is in a stable state;
• alarm has not been treated and is waiting to be treated.

When an alarm is generated, we suppose it requires an intervention. This intervention can be to press a button in response to an event, or to send a team with specialized materials. So with each alarm, the AMM will send the specific information to treat it. Three kind of information can be found within an alarm: one concerning its description, one concerning the action to be taken and one concerning the follow up.

• Description of the event that generated the alarm:
  • Time, date and location of the damage
  • The urban system concerned
  • Description of the alarm

• Action to be taken:
  • Location where the action is to be taken
  • Human and material resources needed
  • Description of the actions to take
  • Validity of the alarm
  • Time to process the alarm

• Alarm’s follow up:
  • State of the alarm (treated, to be treated, etc.)
  • All the actions taken to process that alarm
  • Validity of the alarm
  • Time exceeded since the intervention starts.

Information about the actions to be taken and the alarm’s follow up are dynamic. The AMM interacts with many elements of SIGEC. IDSS and people are the principal ones. IDSS will propose different actions to the AMM based on case-based reasoning. The IDSS, following the previous alarms and actions taken, will find the best action.

![Fig. 4. Alarm control agent and the environment](image-url)
for this particular alarm. The AMM interacts with people, specially an alarm manager who will be able to change the alarm’s information based on the current situations. Fig. 3 shows all the interactions of the AMM within the SIGEC.

We can see that the AMM, while managing all the alarms of the SIGEC, is an element of coordination between each SIDEX' alarm. In that way, the AMM interacts with other elements of the SIGEC (database, UMM, IDSS) to solve a problem in an urban system. It interacts with the IDSS to find similar case. If a case and solution do not exist, the IDSS adds that new case to its knowledge base. The way to solve the problem will be add later in the knowledge base when more expertise from people on the ground will come. The AMM interacts with the User Management Module to contact a responsible who will manage the alarm. The AMM can also interact with others modules of the SIGEC, for example WORK, COMPLAINT, to see if there are some information that can help find the solution to a problem.

Alarms shouldn’t be treated independently because some correlation exists between them. Three types of correlation exist:

- **Spatial correlation:** the AMM has to find if two or more alarms are located in the same area before sending information to a manager;
- **Temporal correlation:** The AMM has to find whether two or more alarms are separated from a small time, this do not implies that the alarms are in the same area;
- **Type correlation:** two or more alarms can be related to the same type of equipment

In that way, the AMM, with the help of other modules can solve a problem in a urban system. Fig. 4 shows the AMM in terms of a MAS. Agent A and Agent B are fixed, they have sensors to detect change in the environment state and they have effectors to modify the environment state. There can be as many fixed agents as there are many different types of alarm. Each kind of alarm will be manage by one type of fixed agent. Those fixed agents can create mobile agent that will find more information, interact with the rest of the environment to find correlation and try to solve the problem.

## 5 SIGEC Specification

SIGEC is a highly parallel system reflecting the real world in the aspects related to the urban infrastructures management where different types of works are done in a simultaneous way, for example, planning, attention to claims, and so on. The system has been completely specified in order to assure the correct performance of all its components. With such specification, the behavior, the inter-relations, the coordination, the cooperation and the communication between the different SIGEC's components are verified in order to develop the system's global and local tasks in a coordinated way.

The language used to formally express the interaction protocols between agents of a multi-agent system, is based on [17]. It uses modal logic operators, world state modeling, actions’ sequentiality and concurrency; additionally, it is based on the speech-acts negotiation model for message communication [16].
The formal model is based on a set of moments with a strict partial order, which denotes temporal precedence. Each moment is associated with a possible state of the world, which is identified by the atomic conditions or propositions that hold at that moment. With each moment are also associated the knowledge and intentions of the different agents. A condition p is said to be achieved when the state is attained in which p holds. A scenario at a moment is any maximal set of moments containing the given moment, and all moments in its future along some particular branch. Thus a scenario is a possible course of events.

A program can be a rule, a pair of sub-programs joined by execution operators, repeating a rule by a certain number of times or until a certain condition is met. A rule is formed by an upper pattern, a lower pattern, and a right side condition. Upper patterns indicate states, conditions, and actions, and if they are matched it is possible to arrive to states and conditions described according to the lower pattern. If the conditions of the lower are met and the actions described on it are accomplished as well, then the condition of the right side of the rule is fulfilled.

An application can be a simple application, an application with a repetition operator analogous to that of rules and programs, or a pair of applications joined by execution operators.

The condition is formed by a logic condition and a state. The logic condition may be expressed through predicate calculus and modal logic, by means of a pair of logic conditions joined by binary operators. The basic condition may be true or empty. When stating that logic conditions may be expressions in the predicate calculus, we are stating that logic conditions can be any well-formed expression with atomic propositions, parentheses, logic operators, quantifiers, that indicates a truth value about the system, in this case a MAS.

The state corresponds to a photograph of the world at a given moment. The state of the model is represented as the global system state along with the specific state of each agent. The global state contains the value of global variables, that is, variables that interest all of the system’s agents, and the state of each agent. If the global state is not relevant, such a set may be empty {}. The state of an agent is a set formed by the internal variables that are relevant to the system. The agent’s migration is atomic. During the agent migration, the agent is an object that migrates between two different nodes, under its own control, and its state does not change. At the end of this process, the agent is in another node and in this moment its state and the global system state change.

Execution operators show how rules, applications and actions are executed. The execution operators are: sequentially, parallelism and the alternative operator. Sequentially: rules, applications, and actions are executed one after an other. Parallelism: rules and applications are evaluated in parallel. The alternative operator allows the execution of rules and applications that satisfy a given condition. In this case, only one is executed.

The logic predicates used in these actions are expressions regarding the agents’ internal state mentioned above. Atomic actions may correspond to changes in the role played within the model. They may also refer to changes in the internal knowledge or to an internal variable relevant to the system or to a change due to the action of another agent or to an internal action defined explicitly in the model. Internal actions are actions executed by an agent that are not seen by other agents and that modify the
state of the agent that executed the action. Finally, these predicates may also refer to message passing.

Below we describe an example that shows the interaction of agents in the model described above. In this example, we present the urban infrastructures’ alarm and control mechanisms. When there is a problem in the infrastructure, the alarm control agent (ACA) creates a mobile agent (MA), which is associated with the alarm. This agent interacts with the geographic information agent (GIA) to know where the problem is, and then with the IDSS agent to find the possible solutions to the problem (Example 1).

**Example 1. Specification 1**

\[
\exists \text{ACA,MG,GIA} \in \text{SMA} \cdot ("\text{Is there any problem in the infrastructure ?"}) \rightarrow ( [\text{interacts with monitoring instruments }]^{\text{ACA}} \cdot [\text{create a mobile agent (MA) with state k}]^{\text{ACA}}\{ \text{<agent: Mobile agent, id: MA,..., state = k, condition problem = "there is a problem P in the infrastructure no-solved">} \}) ;

(MA \rightarrow \text{MSG (}:\text{Content: Request problem-localization(P) : Qualifiers: :}) \rightarrow \text{GIA}) || (MA \rightarrow \text{MSG (}:\text{Content: possible-solution(P) : Qualifiers: :}) \rightarrow \text{IDSS})
\]

\[
[\text{task(localize-problem(P))}]^{\text{GIA}} ||
[\text{Analyze-problem(P)}]^{\text{IDSS}} \{\text{Action B}\}
\]

\[
[\text{B ( k \rightarrow k')}^{\text{MA}}]^{\text{GIA,IDSS}} \{ \text{<agent: Mobile agent, id: MA,..., state = k, condition problem = "there is a problem P in the infrastructure locate and with a possible solution">} \}
\]

6 Implementation Details

The internal structure of an agent is composed of: functionality, knowledge, coordination mechanism, control and communication system. The following section describes some details related to the SIGEC's implementation, specification and functionality.

Functionality refers to the functions or tasks that an agent is able to perform and the other agents can know that it does. The comprises functions such as communication with other agents, obtaining information from the system, information related to the agent's internal status, and so on. An agent's functionality aggregate and its relation with other system's agents have been specified in a formal way by using the language described previously. This language allows us to guarantee the performance of an agent. After the specification related to the agent's behavior as a part of a system, each function must be specified at local level in a more detailed way to allow an implementation with an object-oriented language.

The SIGEC is an abstraction of the real world. Its objective is to operate a urban system. Fig. 5 illustrates the logical data structure of generic SIDEX architecture. It integrates different packages: ORGANIZATION, USER, DATA INTERFACES, INVENTORY, GEOGRAPHICAL IDENTIFICATION, MEASURE and WORK. Some of these packages are located between the SIDEX and SIGEC (IDON, USR, ORG, IGEIO, PLT, TRV). Some data are shared by all the SIDEX and the processing associated with these data is identical from one SIDEX to another. If the processing
associated with the data is different from one SIDEX to another, then they do not need to be at the SIGEC level, but instead at the SIDEX level. The implementation language is JAVA and the mobile agent framework is Grasshopper [10].

The agent development platform used is Grasshopper, which is a mobile platform that is built on top of a distributed processing environment. Grasshopper is implemented 100% in Java and based on international middleware standards (such as CORBA) [10]. Two types of agents act in the Grasshopper context: stationary agents and mobile agents. The intelligent agents are implemented as stationary agents, and the alarm and control agents are implemented as mobile agents.

In open distributed system such as SIGEC a number of serious security threats exist that must be considered when designing an effective security policy. In our system, security has an impact because an urban infrastructure certainly can be an attractive target for sabotage. To address these threats, we use the Grasshopper security services (confidentiality, integrity, authentication, access control and auditing), and also the classic security system such as firewall.

The knowledge communication system is implemented with KQML [6][7][8] (in JAVA), supported over KIF [9], which is used to represent the knowledge or the content of the message itself.

7 Conclusion

In this paper, we proposed and developed managing control and alarms, in an integrated system dedicated to coordinated management of urban infrastructures (SIGEC). This development combines mobile agent technology, and knowledge technology. This combination has been used to achieve a generic system. Our focus on the use of generic models and knowledge representation is the most distinctive feature. The system has a transparent compositional structure based on a generic SIDEX, which it is concerned by the operation of a particular urban system.
addressing questions of its integrated management from the daily operation and preventive maintenance activities to those monitoring its performance and selecting alternatives to improve its response to evolving demands. SIGEC is provided by different measurement and monitoring instruments installed on some system’s elements to be supervised. In this context, mobile agents are used for the real-time management of urban infrastructures’ control mechanisms. To carry out this process, the alarm control agent creates a mobile agent associated with the alarm, which is sent to a mobile station and warns an operator. Preliminary implementation results have shown that SIGEC, supports effectively and efficiently the control and alarm system related to managing urban infrastructures.

Acknowledgments

This work was supported in part by the CERIU and the City of Verdun. A great thank to Guy Leclerc, Grégory Depaix and Philippe Moulin for his ideas and comments.

References

The Spider Model of Agents

Freeman Y. Huang and David B. Skillicorn
Department of Computing and Information Science
Queen’s University, Kingston, Ontario, K7L 3N6 Canada
{huang,skill}@cs.queensu.ca

Abstract. We take the position that large-scale distributed systems are better understood, at all levels, when locality is taken into account. When communication and mobility are clearly separated, it is easier to design, understand, and implement goal-directed agent programs. We present the Spider model of agents to validate our position. Systems contain two kinds of entities: spiders which represent service providers, and arms, which represent goal-directed agents. Communication, however, takes place only between an arm and the spider at which it is currently located. We present both a formal description of the model using the ambient calculus, and a Java-based implementation.

Keywords: agent models, ambient calculus, mobile agents, locality, formal reasoning, Java.

1 Motivation

We present a distributed agent system, called the Spider Agent Model, which is designed to be structurally transparent, both to reasoning and to agent task design, and efficient. The model distinguishes two kinds of entities: spiders, which rarely move and play the role of service providers, and arms, which play the role of agents and are fully mobile.

The design of the spider agent system is motivated by two aspects of existing distributed agent systems which we consider as weaknesses. These are:

1. Most existing systems allow goals to be achieved both by communication and by mobility. When there is only one way to accomplish any goal, it is easier to design the system appropriately, and it is profitable to devote resources to optimize the implementation of the only one possible solution. When there are multiple ways to accomplish a goal, it is hard for users to understand the system, it is hard to choose the best strategy for implementing a particular action, and it is hard to know where best to spend resources to improve performance. A good example is the world wide web which presents, to the user, an illusion of mobility but where all activity is actually implemented by communication. It is hard for a novice user to become more sophisticated; for example, the role of proxies is difficult to comprehend if the user’s mental model assumes browser mobility.
The spider agent system implements information sharing at a distance by mobility, and information sharing locally by communication. These two aspects are clearly different within the system.

2. Existing systems confuse two concepts that we will call virtual mobility and physical mobility. Any distributed system of non-trivial complexity uses virtual names for its objects and a mapping mechanism that associates physical names with them. This mapping need not remain fixed over time.

When an object can move physically, but is still accessible using the same virtual name, then we say that it has physical mobility. Cell phones are physically mobile: their virtual name is their associated phone number, while their physical name depends on the cell they are in at any given moment.

When an object’s virtual name can also change, then we say that it has virtual mobility. For example, when a person moves from one company to another, she exhibits virtual mobility, since all forms of access have changed. Humans handle physical mobility without difficulty, since it does not require changes to our mental maps – only the virtual name needs to be remembered. Virtual mobility is more difficult – human organizations are not constructed to use it, except on very slow time scales. Building artificial systems that are different from human systems is a recipe for opacity at best, and unusability at worst.

Many agent systems confuse these two kinds of mobility, and sometimes go to great pains to implement virtual mobility, which we regard as misguided. For example, ambients do not distinguish between the two kinds of mobility. One ambient can be ‘absorbed’ by another, which is not a behaviour with many direct analogues in the real world.

Of course, the distinction is one of degree not of quality, since it depends on how much work is required to manage redirections. Virtual mobility could be concealed by a further layer of indirection. In the end, the distinction is really whether a name can be resolved to a location in a small constant number of steps (physical mobility) or more (virtual mobility).

In the spider agent model, the service objects (spiders) are physically mobile but not virtually mobile – they maintain the same virtual name throughout.

We show that imposing these limitations on the spider agent system makes it simple to understand and expressive. The extra structure also makes it easier to reason about the behaviour of agents within the system.

The spider agent system is an open architecture in which agents are light-weighted and flexible. Spiders are able to offer arbitrary services, and arms (agents) may contain code that interacts with some or all of the services it encounters.

Section 2 describes some related work. Section 3 describes the spider model in detail. Section 4 introduces two styles of reasoning about programs written using the spider model. The ambient style is discussed in detail. Section 5 describes the prototype implementation.
2 Related Work

Wooldridge and Jennings [8] give four properties for agents: autonomy, agents can act without intervention from outside; reactivity, agents can perceive their environment and act in response; proactivity, agents are goal-directed; and social ability, agents are able to coordinate their strategies with other entities. In a distributed system, one of the actions that an agent can use is to move from one to another processor, making it a mobile agent.

There are many models for agents and agent systems. Useful overviews can be found in [1] and [4]. We highlight three types:

1. Agent systems with CORBA-like goals, that is the ability to assemble components that are physically distributed to make useful wholes. Examples include: Voyager from Objectspace (www.objectspace.com), and Concordia from Mitsubishi Electric (www.meitea.com).
2. Aglets, from IBM Japan, which might best be considered an agent system infrastructure or standard [5].
3. Ambients [3], a general approach to computation in space with a firm semantics.

There are also well-developed systems for reasoning about agents, many based on extensions of standard ways of reasoning in distributed systems. The Ambient model represents a development of ideas from process calculi, and particularly the π-calculus. Such systems are extremely general and powerful, and are often motivated by assuming an environment in which the objects and their actions are constantly changing. This is only a realistic assumption if the environment is considered to include (a substantial part of) the whole system. If communication and mobility are decoupled, then agent actions are associated with a particular location in space. The environment in which they interact is relatively static, altered only by arrivals and departures of other agents, and therefore easier to understand and reason about. An extension of the ambient calculus, called Safe Ambients [6], defines co-action capabilities corresponding to ambient actions for synchronization and interference control of concurrent ambients.

Here is a simple example of ambient interaction synchronized by co-actions. Ambient $k$ in ambient $m$ is equipped with capability to enter ambient $n$ by prior arrangement. It moves from $m$ to $n$ then continues with process $P_k$. Ambients $m$ and $n$ have their own processes $P_m$ and $P_n$ as well:

$$m[\text{out } m. P_m | k[\text{out } m. \text{in } n. P_k)] | n[\text{in } n. P_n]$$
$$\rightarrow m[P_m] | k[\text{in } n. P_k] | n[P_n]$$
$$\rightarrow m[P_m] | n[P_n | k[P_k]]$$

Other approaches to reasoning about agents use temporal [7] or modal approaches to modelling what agents know or believe.
3 The Spider Model

The spider model contains two entities:

1. Spiders, which play the role of service-providing objects. Spiders have unique permanent names, and occur in hierarchies, called spider domains. Spider names, therefore, have the form spider_name@domain_name. Spiders all provide certain basic services related to arm admission, resource allocation, movement and termination. Spiders are typed, and all spiders of a given type provide a known set of services associated publicly with that type. Individual spiders are also free to provide specific services.

2. Arms, which play the role of mobile agents. An arm is created attached to a particular spider (its home spider with which it remains associated in a special way as long as it remains an arm). Arms are free to move to other spider domains (if admission policies permit) where they may make use of the services of the (local) spiders in these domains. Arms may return to their home spider, they may choose to die in any spider domain, or they may choose to settle inside a spider domain and become a new (subsidiary) spider (if policies permit).
   Arms do not have accessible names.

There is a clear separation of mobility and communication. Communication is always local, between an arm and its host spider in the domain where the arm is currently located. Mobility is therefore necessary whenever the information required to meet goals cannot be obtained locally. Note also that communication is always asymmetric, between entities of different kinds, so that agent programs with communication deadlocks cannot be written.

Spiders may also move, but their mobility is incidental to their function. For example, a spider may reside on a laptop. When the laptop is disconnected from the Internet at one location and reconnected at another, the spider has moved, but this has no effect on its behaviour as a spider, nor on any agents currently located in its spider domain. Arms seeking to move to the relocated spider must follow a different path to find it, but this redirection is really a function of the underlying network.

Allowing arms to communicate only through intermediary actions of spiders imposes structure on the patterns of actions of agents. On the one hand, this is a significant limitation, since emergent complex behaviour based on the interaction of many simple agents is harder to express. On the other hand, it does permit computations to exploit locality – and agent need only be prepared for what it might encounter within each spider domain at a time, not for everything it might encounter in the entire system. As the “entire system” increasingly becomes an entity that spans the globe, this is a major saving in complexity, both intellectual during design of the agent, and performance by reducing the amount of code an agent must carry against contingencies. Enforcing communication via spiders also means that resources can be held in the static pieces of the system (the spiders) rather than carried around in the mobile pieces (the arms).
Separating communication and mobility enforces a data-centric view of a distributed computation, in which code moves towards data rather than the other way around. This makes better use of network bandwidth, which may be important if parts of the network are wireless.

3.1 Spiders

A spider domain consists of a hierarchy of spiders, each of which is interacting with a set of arms. These arms are of two kinds: the spider’s own arms which it has created, typically in response to a request from a user (located ‘at’ this spider domain) that requires accessing remote data; and arms from other spiders that are currently located at this spider (‘just visiting’).

Spiders admit visiting arms based on their security policies. An arm admitted into a spider domain must initially ask for one of three things: to die, to be moved to some other spider domain, or to be given a set of resources. Whether, and how much of these resources are granted by the spider depends on its local policies, but it is important that all types of resources are requested and granted atomically to prevent resource deadlocks within spiders. In the prototype, the only resource considered is computation cycles.

A spider provides a resource-bounded playground for each arm in which it may consume the allocated resources in any way it wishes. However, this typically involves interactions with the spider invoking any of the services that this spider provides.

Spiders may know the names of other spiders, and may reveal these to arms as a service. Requests to move are at the instigation of arms, which must know the name of the spider to which they wish to move. However, wild cards in names are possible, in which case the current spider is free to move the arm to any matching destination spider. Including wild cards enables arms to access services without having to know the names of individual spiders.

3.2 Arms

As described in the previous section, an arm arriving in a spider domain typically begins by asking for a grant of resources. After this, it may interact with the spider using any of the following actions:

- Reconfiguration. Spiders guarantee to provide arm code for certain generic parts of arm structure, and may also contain certain kinds of type-specific arm code. Hence an arm only needs to carry (a) enough code to gain access to a spider domain, and (b) code specific to its mission, since it can pick up other code inside spider domains. This can be implemented using Java’s mechanisms to include code from packages at different locations.
- Standard services. These include:
  - Die. Remove this agent from existence inside the spider domain.
  - Move. This requires a spider name to be given as an argument. The arm is repackaged for movement and sent to the given destination.
• Settle. The arm is given standard spider code and becomes a descendant of the current spider in the hierarchy, if this is permitted. If not, control returns to the arm to take appropriate action.

- Specialized spider services. Spiders of the same class offer services standard to that class. An individual spider may also offer particular services. The interface for services is generic, indexed by an unbounded service number. Information about which spiders offer which services is ordinary data and accessible in ordinary ways.

The necessary parts of the spider model have been kept as small as possible. Useful systems require more than this basic set of services. For example, a spider name service, search engine spiders, and spiders that maintain a persistent public storage area (e.g. in the style of the MARS project [4]) are all likely to be common extensions. Notice that many standard web-based activities are naturally implementable using the spider model, with the important difference that browsing, search, and so on actually use mobility. Applications such as cooperative search require spiders with public persistent storage. Cooperating arms can use such storage to communicate with each other – but note that communication is spider-mediated and asynchronous making it much harder to create trivial deadlocks.

<table>
<thead>
<tr>
<th>Table 1. Definition of Arm</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a, b$</td>
</tr>
<tr>
<td>$A, B :=$</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>$A</td>
</tr>
<tr>
<td>$E.A$</td>
</tr>
<tr>
<td>$E :=$</td>
</tr>
<tr>
<td>$C$</td>
</tr>
<tr>
<td>$R :=$</td>
</tr>
<tr>
<td>$moveto(s)$</td>
</tr>
<tr>
<td>$die$</td>
</tr>
<tr>
<td>$reqres(q)$</td>
</tr>
<tr>
<td>$settle(s)$</td>
</tr>
<tr>
<td>$getserv(p)$</td>
</tr>
<tr>
<td>$q$</td>
</tr>
<tr>
<td>$p$</td>
</tr>
</tbody>
</table>

4 Reasoning about the Spider Model

The two entity types of the model are defined in Tables 1 and 2. In Table 1 it is supposed that $q \in QoR$ where $QoR$ is a set of values on which a partial order,
addition, subtraction and 0 are well defined. Definition of QoR is application-dependent. \( rm_a \) in Table 2 is the resource manager for arm \( a \) created by its host spider, responsible for resource approval, deduction of consumed resource, and termination of arm \( a \) when it runs out of resource.

There are four types of services provided by each spider, corresponding to the four basic types of service request arm actions. Other services can be provided and requested through a general parametric interface \( service \), which is linked to specific services that differ from spider to spider. A spider can contain child spiders as well as arms. Arms can only be active within spider and can communicate only with their host spiders. The interactions between arms and spiders are represented by the reduction rules in Table 3. (The convention of substitution notation is adopted: \( A\{x \leftarrow value\} \) means substitution of all free occurrences of variable \( x \) in process \( A \) by \( value \).

\( q_{min} \) in R-mov is an initial amount of resource that is just enough for the incoming arm to request further resource or to move to somewhere else if local resources are not granted. Value \( q_1 \) in R-auth is the granted resource based on the requested amount \( q_1 \), the spider’s resource situation, and the resource management policy. Note that the request action itself consumes some resource \( q_{rr} \) (\( q_{rr} < q_{min} \)). Value \( r_p \) in R-serv is the returned result from the requested service (suppose that \( A \) needs the result for variable \( x \)). Services related to interface \( service \) can be identified and invoked by their IDs or names included in the parameter list.
The spider is illustrated in Figure 1. Weight because they just call corresponding spider services. The architecture of actions described in Table 1 for programmers to invoke. These methods are lightweight and a pointer to the current host spider, and has methods representing basic mobility based on Java object serialization. A general arm holds an unique ID from spider.

A prototype of the Spider Model has been implemented using Java, with arm mobility based on Java object serialization. A general arm holds an unique ID with services $V_1$ and $V_2$ (assuming that security and resource policies permit it).

For all the reduction rules except R-exht and R-die, we assume that resource $q$ is sufficient for the requested service. If not, the corresponding reduction cannot be completed and the arm will be killed in-process by R-exht.

A simple but typical application of the reduction rules is an arm $a$ moving from spider $s$ to $t$ then asking for resource $q_{set}$ to settle down as spider $s_a$ with services $V_1$ and $V_2$ (assuming that security and resource policies permit it).

5 Implementation

A prototype of the Spider Model has been implemented using Java, with arm mobility based on Java object serialization. A general arm holds an unique ID and a pointer to the current host spider, and has methods representing basic actions described in Table 1 for programmers to invoke. These methods are lightweight because they just call corresponding spider services. The architecture of a spider is illustrated in Figure 1.

First of all, each spider executes a Receiver on a particular port number. The Receiver is responsible for receiving incoming arm code, authenticating for admission, recreating the admitted arm from code and allocating initial amount of resource for it. If the arm is rejected, the sender spider is notified. Complianted systems can have a separate security manager to maintain a sophisticated security policy.
Mobility service consists of receiving and sending services. When an arm requests move service, the host spider will contact the receiver of the destination spider at the particular port.

Another important part of a spider is the resource manager which maintains a resource policy, deals with resource requests from residing arms, and monitors resource consumption. Note that although many other models and systems use the phrase resource manager, they usually mean a data storage/allocation manager, a completely different concept.

The spider services are invoked by arms through corresponding public method calls. These are the only way for arms to interact with their environment. Spiders keep local pointers to residing arms private. To show the simplicity of programming using the spider model, the code of a specific arm that explores a set of spiders is shown below. It is given an initial itinerary. When it arrives at a new spider, it first spends some time doing local work (doMyTask), and then moves to the next spider currently at the front of its itinerary.

```java
public class ExplorerArm extends Arm {
    public static final int REQUIRED_CPUTIME = 5000; // in ms
    public static final int NUMBER_OF_STOPS = 10;
    private int requiredCPUTime = REQUIRED_CPUTIME;
    private Vector itinerary = new Vector(NUMBER_OF_STOPS);
    public ExplorerArm(Spider hostSpider, AgentID myID) {
        ...... // given the itinerary
    }
    public void run() {
        if (itinerary.isEmpty()) {
            System.out.println(this + " finished task, dying...");
            die();
        }
        if (! showID().getOwner().equals(getHostLocation()))
        { Resource grantedRes = requestResource(new Resource(requiredCPUTime));
```
if (grantedRes.compareTo(new Resource(requiredCPUTime)) >= 0)
  doMyTask();
}
String destName = null;
while (!itinerary.isEmpty())
{
  destName = (String) itinerary.firstElement();
  itinerary.remove(destName);
  moveMeTo(new SpiderAddress(destName, Spider.DEFAULT_PORT));
  System.out.println("No spider available at host "+destName);
}
//no more spider hosts available, go home.
if (!moveMeTo(showID().getOwner()))
{
  System.out.println(this + " becomes homeless, dying...");
  die();
}
protected void doMyTask()
{
  ......;
}

This code is very stylized. A much more abstract language, in which agent actions were described at the level of “move there”, “search for this” can be straightforwardly mapped (compiled) to such code.

6 Conclusions

Our position is that large-scale distributed systems are better understood, at all levels, when locality is taken into account. It is more natural, more efficient, and easier to reason when the concepts of communication and mobility are clearly separated and clearly visible in the model.

To support this position, we have designed and implemented the spider agent model. The distinguishing features of the model are:

- Two kinds of entities: spiders, which represent service providers, and arms, which represent goal-fulfilling distributed computations.
- Insistence that communication can only take place locally (that is, within a spider domain), so that there is only one way to acquire remote information — by moving to the location where it exists. Hence, mobility is not an extra, optional feature, but a necessity.
- Because of restrictions on form, there is typically only one way to achieve any particular goal. This helps with design clarity, and also directs attention to those system aspects that most repay optimization.
- Because of the restrictions on form, reasoning about program behaviour is simplified.
References

On the Modeling of Mobile Agent-Based Systems

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

Laboratoire d’Informatique de Paris 6,-8, rue du Capitaine Scott— F 75015 PARIS
* University Paris 6 ** University Paris X
tel: +33.1.44.27.43.63 fax: +33.1.44.27.74.95

Abstract. The mobile agent technology provides facilities that enable to reduce the complexity of telecommunication services development. However the major part of this development is still devoted to the code production. In order to optimize the development of services, this production should be reduced while the main part must become an upstream activity, i.e., the elaboration of specifications. This paper introduces an Architecture Description Language (ADL) devoted to the design of mobile agent systems to be implemented in a MASIF compliant platform. The ADL is defined as a UML profile called the MASIF-DESIGN profile. It enables the designer to describe the platform he/she uses, to locate the agents in the platform and to define the elements required from the platform for the achievement of the distribution transparencies.

1 Introduction

The demand for sophisticated telecommunication services drastically increases inducing a growing complexity of these services. Moreover, the reduction for time-to-market is one of the competitive criteria for the providers who consequently need powerful tools to assist them in the rapid introduction of services. In particular, it is commonly considered that a good way to obtain quickly new services is to adapt existing ones by adding new functionalities.

In this context of software reuse, the mobile agent technology facilitates this introduction of new services since agents are software entities easily adaptable. Thus, a service can be seen as a set of interacting agents. Developing a new service can consist for example in modifying the internal behavior of an agent, modifying some interactions between agents or adding new agents that will interact with the existing ones. Today, standards related to mobile agents become available and platforms compliant to these standards and supporting mobile agent-based systems are available. They enable the construction of services as a set of agents that are able to move in order to achieve their goals. For that, they provide all the needed mechanisms for the execution of agents. They offer APIs used by the developer when coding agents. These APIs make available the platform services for the agents when running. These platforms provide abstractions that ease the construction of mobile agent-based systems by hiding technical aspects such as access and location, security,
migration or communication over heterogeneous networks. Thanks to these platforms, the developer can focus on the functional part of the system being developed, i.e., the business model, rather than on the non-functional part, i.e., the technical and implementation aspects taken into account by the platform services.

Although the mobile agent technology provides facilities that enable to reduce the complexity of services development, however the major part of this development is still devoted to the code production, even if it is based on code reuse. Now it is recognized that to be really efficient, services development must be thought in terms of specification production and reuse rather than in terms of code production and reuse. Let us see for example the recent OMG standard, namely the Model Driven Architecture (MDA) that recommends the use of modeling techniques as a way to simplify the construction of applications [1]. In this way, the main part of the development becomes an activity upstream from coding, namely the analysis and design dedicated to the elaboration of the application specification. This results from the composition of existing pieces of specifications. New pieces of specifications can be elaborated only when it is needed. Most of the code is then generated and the code production becomes the smallest part of the work.

Based on these considerations, LIP6 started a project named ODAC (Open Distributed Applications Construction) that aims to provide a methodology to develop distributed applications. The methodology is general enough to be adapted to any kind of applications. Nevertheless we focus on several application domains such as mobile agents systems [2]. A methodology must define a set of concepts, the usage rules of these concepts by organizing them into various steps, the process associated with these steps and a notation. ODAC makes use of the Reference Model of Open Distributed Processing (RM-ODP) concepts. RM-ODP is an ISO and ITU-T standard related to the distributed processing [3]. This defines a set of rigorous concepts for modeling distributed systems. It makes use of the object paradigm in such a general way that it is possible to deal with ODP objects in the same way as they would be agents. In our view, an agent is an ODP object and mobile agent-based systems are distributed in the ODP term sense, since they comply in a technical and organizational heterogeneous context. They consist of interacting entities, which can be agents and/or objects. Thus ODAC lies within the ODP standard scope. This allows specifying mobile agent-based systems according to the ODP semantics. We associate the use of the UML standardized notation, which allows the specifications writing. Thus, we are defining an Architecture Description Language (ADL) for modeling the system both in analysis and design phases. In a first step of the ODAC methodology development, we have defined the part of the ADL devoted to the analysis phase.

This paper focuses on the design phase of the ODAC methodology and the part of the ADL we are defining for it. We first provide in Section 2 some background on ODAC needed to tackle the Sections 3 and 4 in which we detail the ADL part devoted to the design of a mobile agent system. The use of this ADL is illustrated in Section 5.

2 Background on the ODAC Methodology

As mentioned previously, when developing a telecommunication service, the developer must focus on the functional part of the system, namely the business model. This part does not depend on the target environment in which the system will run.
Once it is described, then the developer must take into account the non-functional part of the system that depends on the technical environment in which the system will be implemented.

According to these two parts, the ODAC methodology identifies two kinds of specifications. The \textit{behavioral specification} describes the system according to its objective, its place in the company in which it is developed, information that it handles and the tasks that it carries out. It corresponds to the functional part of the system described in the analysis phase. According to the ODP separation of concerns, the behavioral specification results from the specifications established in the Enterprise, Information and Computational viewpoints. To express a behavioral specification, we provide the modeler with an ADL based on the UML standard notation. We then have mapped the RM-ODP concepts of the three mentioned viewpoints onto the UML ones [4].

The \textit{operational specification} results from the design step corresponding to the projection of the behavioral specification on a target environment reflecting the real execution environment. It constitutes the description from which code is generated and the implementation is carried out. According to the ODP viewpoint concept, it depends on the specification established in the Engineering viewpoint, which describes the execution environment. We have then supplemented our ADL used in the analysis step in order to include the design concerns. We give hereafter an overview of the first version of this ADL we defined [5].

The ADL deals with the environmental infrastructure issues involved in a mobile agent system specification. Thus it includes concepts related to the distribution aspects while enabling the description of the considered environment. It is defined as a UML profile. This is called “MASIF-DESIGN” profile, as for now, the distributed execution environment we consider is in conformance with the OMG-MASIF standard. MASIF presents a minimum set of concepts and operation interfaces necessary for interoperability. The term operation in this context has a UML meaning. The operation is the function equivalent in an ODP Engineering viewpoint context. Defining an ADL in order that a designer of mobile agent systems can describe an operational specification requires the consideration of two issues, namely the description of the considered environment and the representation of distribution transparencies.

A MASIF compliant agent environment considers the following platform elements: Region, AgentSystem, CoreAgency, Place and Agent. The MASIF-DESIGN profile provides the corresponding UML representation of these platform elements as illustrated in Table 1.

A distribution transparency hides aspects related to the distribution in the behavioral specification, assuming existing mechanisms that will detail these aspects in the operational specification. Actually, the platform elements cooperate to provide a transparency by bringing uniformity to some aspects of agents’ distribution (e.g. uniformity of naming whatever the location of the agent). The transparencies have to be specified as analysis phase requirements. They enable to refine the existing behavioral specification with introducing additional behavior, including the use of one or more operations of the platform elements. We provide in the MASIF-DESIGN profile some tagged values such as location or authority that enable the designer to specify parts of location and access transparencies.
Table 1. The MASIF-DESIGN profile part for the modeling of the MASIF platform elements

<table>
<thead>
<tr>
<th>MASIF platform elements</th>
<th>UML Meta-model Class</th>
<th>Stereotype Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region</td>
<td>Stereotyped Subsystem</td>
<td>Region</td>
</tr>
<tr>
<td>Agent System</td>
<td>Stereotyped Node</td>
<td>Agent System</td>
</tr>
<tr>
<td>Core Agency</td>
<td>Stereotyped Subsystem</td>
<td>CoreAgency</td>
</tr>
<tr>
<td>Place</td>
<td>Stereotyped Package</td>
<td>Place</td>
</tr>
<tr>
<td>Agent</td>
<td>Stereotyped Component</td>
<td>Agent</td>
</tr>
</tbody>
</table>

Thus the MASIF-DESIGN profile provides a way to write an operational specification when the considered target environment is a MASIF compliant platform. However, the MASIF standard does not describe in details the mechanisms achieving the distribution transparencies. The set of operations presented in the standard offers some limited possibilities regarding these mechanisms. Actually, these must be defined by the platform providers. For example, Grasshopper that is a MASIF compliant platform implements operations that permit to deal with the agent execution environment concerns [9].

Studying such an example enabled us to enhance our MASIF-DESIGN profile by adding some elements not described in the MASIF standard but needed for the modeling of mobile agent platforms. We present hereafter the updated version both for the modeling of the platform elements and for the representation of the distribution transparencies.

3 Modeling of the Platform Elements

As mentioned previously, a MASIF compliant agent environment considers some platform elements such as Region, AgentSystem, CoreAgency, Place and Agent. Each platform element offers a set of operations that represents the implemented interactions between the platform elements. These interactions form the refinement needed for a behavioral specification, refinement that details the transparencies in the operational specification of an agent system.

The Region is a registration facility supporting localization. We model it as a stereotyped subsystem, considering a subsystem as in [7]. Besides the MAFFinder interface specified in MASIF, two more groups of operations can be provided in order to facilitate Agent Systems domain services. For example, a platform such as Grasshopper provides the lookupCommunicationServer() operation that permits to know the underlying communication mechanism that agents of an AgentSystem use (e.g., socket, CORBA or RMI). These operations are implemented in methods with the prototype defined in the interface IRegion (Fig. 1). The complete list of functionalities can be found in [9].
The **AgentSystem** is the platform that can create, interpret, execute, transfer and terminate agents. We represent it as a stereotyped node. Each agent system has one **CoreAgency** and one or several places. The AgentSystem acts as a container for executing agents, and the functionalities of an AgentSystem are provided by the **CoreAgency**.

The **CoreAgency** implements the agent execution management for an **AgentSystem**. We model the **CoreAgency** as a stereotyped subsystem. In addition to the operations identified in the **MAFAgentSystem** interface defined in the MASIF standard, other operations can be identified that permit to monitor and control locally running agents. Then we add the **IAgentSystem** interface such as defined in Grasshopper (Fig. 2). An example of operations that can be found in this interface is the `saveAgent()` operation that saves the agent data for a future restoring. The complete operations list can be seen in [9].

**Fig. 1**: Region stereotyped subsystem

**Fig. 2**: CoreAgency stereotyped subsystem
A Place is a context within an Agent System in which an agent can run. We model the Place as a package. It can provide functions such as access control. The designer can define in an interface named SpecialPlaceInterface the operations that offer services to agents located in a SpecialPlace. There can be no more than one SpecialPlace in an AgentSystem (Fig. 3).

We represent an Agent as a stereotyped component. Additional information needed for the implementation can be included in the component diagram related to an agent implementation. Once again, we use the Grasshopper example to identify this additional information. Since Grasshopper is a typed agent environment, the Agent implementation modeled as a component has to inherit a specific structure [10]. This forces the designer to consider the special operations for the design. For example, the move() operation permits the agent migration. The beforeMove() and afterMove() operations permit to prepare the agent for the migration (i.e. save the execution state) (Fig. 4).

4 Representation of the Distribution Transparencies

A distribution transparency is the capacity of hiding the distribution aspects in the behavioral specification. RM-ODP defines a set of transparencies needed for a distributed system, which are: access transparency, failure transparency, location transparency, migration transparency, relocation transparency, replication transparency, persistence transparency and transaction transparency [3].

We focus on the access transparency, location transparency, persistence transparency, relocation transparency and migration transparency by identifying for
each of them the tagged values enabling to define them (Table 2). In addition, operations can be defined that contribute to achieve transparencies.

4.1 Access Transparency

Access transparency masks differences in data representation and invocation mechanisms to enable interactions between agents. Here, there are two issues to consider, namely the establishment of the communication link and the security aspects in terms of access rights.

The link establishment is based on mechanisms such as RMI, IIOP or socket that are initialized for each AgentSystem. In order that the designer can specify which communication mechanism he/she wants to use, we define the tagged value linkInfrastructure for each agent. Through this communication link, an agent can use operations from another agent. Thus the designer must decide and define the operations that an agent makes available for other agents. In fact, among the set of operations of an agent, he/she decides which operations can be accessed by other agents and places the signature of these operations in an interface called IExported<agent_name>Operations (e.g., IexportedAgent1Operations). Each of these operations represents an interface of the component that implements the agent. The designer can specify these operations in one way or the other as illustrated in Fig. 5a or 5b.

Considering this example, another agent Agent2 can interact with the Agent1 only through the operations regrouped in IExportedAgent1Operations.

![Fig. 5: Agent accessible operations specification](image)

The second issue that has to be considered for access transparency is the security and the rights to access the agent.

Security applies to communication. Platforms provide some mechanisms ensuring secured communication, such as rmissl or socketssl. So we define a tagged value named securityProtocol that the designer can initialize in order to specify which secured protocol he/she wants to use.

An agent reaches the operations made available from another agent if it has the corresponding access rights. These rights are related to a defined policy. The policy can be activated or not for every AgentSystem. In order that the designer specifies the activation of the policy, we define a tagged value named policyApplied. The policy is defined in a special file with the name and location that the designer can specify thanks to the tagged value policyFile. The policy is applied in correlation with the agent’s Authority. This Authority identifies the person or organization for whom the agent acts and for this, we define the tagged value Authority. In this way, an Authority must be authenticated at each communication access based on the AgentSystem policies.
4.2 Location Transparency

Location transparency masks the use of information about location in space when identifying and binding interfaces of agents. Thus, agents can interact with other agents without using the location information. There are two location issues, namely the location of an agent and the location of resources files for an agent.

In MASIF standard and MASIF compliant environments, the location is defined as the path to an agent system based on the AgentSystem, the agent or the place. The operation named `lookup_agent()` returns the location of an agent. This operation permits to connect two agents without knowing their locations by requesting the region about the agents’ location. Nevertheless the designer can specify the location of the agent in an agent tagged value named `Location`. In this way, the designer can specify a changed location for an agent.

We have also to consider the location of the agent definition file. An AgentSystem that executes an instance of an agent needs this agent definition. Based on the designer specification of the location of this file, some operations like `MAFAgentSystem.fetch_class()` permit afterward to retrieve this definition. We provide a tagged value `FileDefinitionLocation` in order that the designer specifies this location.

4.3 Persistence Transparency

Persistence transparency masks the deactivation and reactivation of the agents. In particular, it masks the deactivation imposed by specific constraints of processing, storage and communication (e.g. the agent deactivation if it is not accessed for a specified period of time in order to save processing capacity). Agents can be persistent or not. The designer has to choose which agents will be persistent and which ones will not be. Thus we define a tagged value named `Persistent` that enables the designer to specify this. In the same way, the designer has to specify if the persistency service of an AgentSystem is enabled or not into the tagged value `SystemPersistencyEnabled`.

Mobile agent platforms support agents persistency by providing services in terms of available operations. When a designer writes an operational specification, he/she can use them. Some of them relate to the agents deactivation while others relate to the agents saving.

Generally, two ways are provided by platforms to deactivate agents. One is to explicitly invoke commands like `deactivate()` or `flush()` available for an agent as a part of the `AgentsOperations` interface. Some of these operations are also available for the CoreAgency. The `flush()` operation not only deactivates the agent but also deletes it from the AgentSystem. Another way is to specify a certain amount of time. When this amount of time passed since the last access to that agent, it is automatically deactivated and deleted from the AgentSystem. In order that the designer can specify this amount of time, we define the tagged value named `AgentTimeout`.

In addition, operations are provided by the environment for the designer like `beforeflush()` and `afterLoad()` in order he/she uses a state saving mechanism. Actually, only the signatures are provided, as part of the `AgentsOperations` interface. If the designer wants to use an execution state saving mechanism then he/she has to define
it explicitly in the core parts of these operations. They are called automatically, in a transparent way for the designer, before and respectively after a `flush()` operation.

Besides the transparent saving procedure of an agent, the designer can explicitly specify the saving of the agent by defining the invocation points of the provided operation `save()`. This operation is part of the `AgentOperations` interface but there is also a similar operation named `saveAgent()` in the `IAgentSystem` interface that the `CoreAgency` implements.

### 4.4 Relocation Transparency

*Relocation transparency* masks the relocation of an agent from other agents that communicate with it. The designer specifies the operations that an agent is making available for other agents as shown in Section 4.1. Besides the definition of the `IExported<agentname>Operations` interface, the designer has to register the `AgentSystems` involved in the same `Region`. This specification and the access transparency tagged values configuration are sufficient for the designer in order that the execution environment offers afterwards a transparent implementation of this transparency.

### 4.5 Migration Transparency

*Migration transparency* masks the ability of the platform to change the location of an agent. The migration of agent’s data is assured transparently thanks to the invocation points of the `move()` operation defined by the designer. The signature of this operation is also in the `AgentsOperations` interface and its core part is implemented by the environment. Like for the persistence transparency, the designer can detail an execution state saving mechanism in `beforeMove()` and `afterMove()` operations when the environment does not provide it.

As we have said, the agent data is transparently migrated with the agent. However, the designer can specify if some data don’t have to be migrated. For this, he/she has to tag the data with `transient` keyword when he/she defines it in the Information viewpoint.

### 4.6 Summary

All the enumerated operations needed for ensuring transparencies are included in our MASIF-DESIGN profile definition. Tables 2a and 2b summarize all the operations and the tagged values defined in the profile to support the transparencies definition during the design phase.
Table 2a: Operations of the MASIF-DESIGN profile for transparencies

<table>
<thead>
<tr>
<th>Transparence</th>
<th>Operation</th>
<th>Interface</th>
<th>Platform Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>LookupCommunicationServer()</td>
<td>IRegion</td>
<td>Region</td>
</tr>
<tr>
<td>Location</td>
<td>GetMAFFinder()</td>
<td>MAFAgentSystem</td>
<td>CoreAgency</td>
</tr>
<tr>
<td></td>
<td>Fetch_class()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lookup_agent()</td>
<td>MAFFinder</td>
<td>Region</td>
</tr>
<tr>
<td>Persistence</td>
<td>Deactivate()</td>
<td>AgentOperations</td>
<td>Agent</td>
</tr>
<tr>
<td></td>
<td>Flush()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Save()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SaveAgent()</td>
<td>IAgentSystem</td>
<td>CoreAgency</td>
</tr>
<tr>
<td></td>
<td>ReloadAgent()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relocation</td>
<td>Register_agent_system()</td>
<td>MAFFinder</td>
<td>Region</td>
</tr>
<tr>
<td></td>
<td>Unregister_agent_system()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Migration</td>
<td>MoveAgent()</td>
<td>IAgentSystem</td>
<td>CoreAgency</td>
</tr>
<tr>
<td></td>
<td>Move()</td>
<td>AgentOperations</td>
<td>Agent</td>
</tr>
</tbody>
</table>

Table 2b: Tagged values of the MASIF-DESIGN profile for transparencies

<table>
<thead>
<tr>
<th>Transparency</th>
<th>Tagged Value</th>
<th>Platform element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>LinkInfrastructure</td>
<td>Agent, AgentSystem</td>
</tr>
<tr>
<td></td>
<td>SecurityProtocol</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PolicyApplied</td>
<td>AgentSystem</td>
</tr>
<tr>
<td></td>
<td>PolicyFile</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Authority</td>
<td>Agent</td>
</tr>
<tr>
<td>Location</td>
<td>Location</td>
<td>Agent</td>
</tr>
<tr>
<td></td>
<td>FileDefinitionLocation</td>
<td></td>
</tr>
<tr>
<td>Persistence</td>
<td>Persistent</td>
<td>Agent</td>
</tr>
<tr>
<td></td>
<td>AgentTimeout</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RepeatedSaveTimeout</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SystemPersistencyEnabled</td>
<td>AgentSystem</td>
</tr>
</tbody>
</table>

5 Case Study: The Travel Agency

We illustrate in this Section how a designer makes use of the MASIF-DESIGN profile in order to write the operational specification of the application he/she is developing, namely an electronic travel agency. This example comes from the FIPA specifications [8]. Customers represented by their Personal Travel Assistant PTA buy travels near an agency called thereafter “Travel Broker Agent” (TBA). This TBA is in contact with travel service companies (e.g., transport companies, hotels, etc.) that are called Travel Service Agent (TSA). It acts as intermediary between the PTA and the TSAs. Here, we are focusing on the distribution issues and transparencies descriptions that have to refine a previous behavioral specification.

5.1 Model of the Electronic Travel Agency

Thanks to the various stereotypes of the MASIF-DESIGN profile, the designer can define and represent his/her environment. For example, he/she chooses to have a Region Voyage in which two AgentSystems, namely Organizer and TravelerProvider are registered. In each of these AgentSystems, he/she creates only one place named
InformationDesk. Each of these platform elements is modeled by using the corresponding MASIF-DESIGN profile representation. For sake of simplicity, we only provide the deployment diagram that illustrates the location of the various agents in this model (Fig. 6)[5].

![Deployment Diagram](image)

**Fig. 6.** The deployment diagram for the agents of the Voyage travel agency domain

In the Organizer AgentSystem, the designer chooses to locate the PTA and the TBA. In this way, they can locally interact for example by operation calls. To represent the migration of the TBA from the Organizer AgentSystem to the TravelerProvider AgentSystem, the designer uses the `become` UML stereotype. This enables to present the TBA in both AgentSystems according to the fact that it resides in both AgentSystems during its lifetime. Once the TBA resides in the TravelerProvider AgentSystem, it can interact with the TSA in the same way as PTA and TBA interact in the Organizer AgentSystem.

### 5.2 Representation of Distribution Transparencies

#### 5.2.1 Access Transparency

The designer must define the value of each tagged values involved in the achievement of the access transparency by the environment. We summarize in Table 3 an example of the design for the electronic travel agency. Tagged values must be configured for each platform element they apply (cf. Table 2).

---

1 In this case study, since we are not considering behavioral issues like message passing though the different communication techniques (e.g. asynchronous communication), we choose this minimal method to achieve communication.
In addition, the designer can define the operations that contribute to achieve access transparency. These operations represent the available services and their signatures are in an interface attached to each agent (Fig. 7).

The operation `SendForInformationRetrieval` is accessed by the PTA when it requests the TBA for a travel. The TBA accesses the operation `AskForInformation` in order to obtain the information from the TSA. The operation `AcceptInformation` is accessed by the TSA when it responds with the information to the TBA. The operation `SetRetreatedInformation` is accessed by the TBA when it comes back towards the PTA with the information.

### Table 3: Tagged values of the access transparency

<table>
<thead>
<tr>
<th>Tagged Value</th>
<th>Organizer</th>
<th>Travel Provider</th>
<th>PTA</th>
<th>TBA</th>
<th>TSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinkInfrastructure</td>
<td>Socket</td>
<td>Socket</td>
<td>Socket</td>
<td>Socket</td>
<td>Socket</td>
</tr>
<tr>
<td>SecurityProtocol</td>
<td>socketssl</td>
<td>socketssl</td>
<td>socketssl</td>
<td>socketssl</td>
<td>socketssl</td>
</tr>
<tr>
<td>PolicyApplied</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authority</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>globalAuthority</td>
</tr>
</tbody>
</table>

**Table 4.** Tagged values of the location transparency

<table>
<thead>
<tr>
<th>Tagged Value</th>
<th>PTA</th>
<th>TBA</th>
<th>TSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Organizer/InformationDesk</td>
<td>Organizer/InformationDesk</td>
<td>TravelProvider/InformationDesk</td>
</tr>
</tbody>
</table>

5.2.2 Location Transparency

The designer must define the value of each tagged value involved in the achievement of the location transparency by the environment. We summarize in Table 4 an example of the design for the electronic travel agency. Tagged values must be configured for each platform element they apply (cf. Table 2).
The *FileDefinitionLocation* is used as a parameter by the environment for the creation and the migration to retrieve the definition of an agent.

Before migration, the TBA must locate the TSA. It first uses the operation `getMAFFinder()` of the CoreAgency of the Organizer AgentSystem to find the *Voyage* region, then its calls the operation `lookup_agent()` of the *Voyage* region (cf. Section 4.2). When it comes back to its original location, it already knows it and finds it transparently.

### 5.2.3 Persistence Transparency

Tagged values configured by the designer for the achievement of the persistence transparency are summarized in Table 5.

<table>
<thead>
<tr>
<th>Tagged Value</th>
<th>Organizer</th>
<th>Travel Provider</th>
<th>PTA</th>
<th>TBA</th>
<th>TSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persistent</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>AgentTimeout</td>
<td>36000ms</td>
<td>36000ms</td>
<td>36000ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RepeatedSaveTime</td>
<td>1800ms</td>
<td>1800ms</td>
<td>1800ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SystemPersistencyEnabled</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Concerning the operations, we have already mentioned in Section 4.3 the `flush()` operation. This enables to remove agents from their AgentSystems for resource saving reasons. For example, TBA and the TSA are deactivated and removed after a full execution cycle. Their data are saved and will be restored at the reactivation when the agent will restart. The reactivation is done transparently by the environment when an agent reaches the operations of another one. For example, the TBA will be reactivated when the PTA will request a travel and the TSA will be reactivated when the TBA will access it. In some cases, the reactivation can be done on request.

In the current version of the MASIF-DESIGN profile, we do not provide the designer with means to express the execution’s state saving related to the deactivation and removal of an agent. However, as explained in Section 4.3, the designer is free to specify the core part of the two operations `beforeFlush()` and `AfterLoad()` in order that an execution state saving mechanism will be available, similar with the one described late further for the migration transparency.

### 5.2.4 Relocation Transparency

By providing the UML diagrams as illustrated in Fig. 7, the designer specified the operations through which the three agents interact. Based on this specification, the interfaces `IExportedPTAOperations`, `IExportedTBAOperations`, `IExportedTSAOperations` are defined. This specification and the access transparency tagged values are sufficient and permit afterwards the creation of a full implemented mechanism that ensures the achievement of the relocation transparency.

### 5.2.5 Migration Transparency

The `Move()` operation of the `AgentsOperations` interface enables the agent migration. This refers to data migration. In our example, according to our location choices for the agents (see Fig. 6), only the TBA migrates to interact with a TSA. Thus data to be migrated are data that the TBA retrieves from the TSA. As mentioned previously, a
designer can choose to have a mechanism that ensures the state migration. We provide here an example of the description realized by a designer when he/she wants to specify a TBA state migration.

Let us assume that the TBA behavioral specification can be separated by the designer in three composing parts (cf. Section 5.2.1). This separation is a consequence of the designer choice to mark the points in the specification of the agent behavior when a migration takes place:

- A set of actions performed by the TBA when it resides on the Organizer AgentSystem and waits a call from the PTA. This call is the local interaction SendInformationRetrieval() between the PTA and the TBA. It triggers the next part of the TBA behavior, namely

- The interaction between the TBA and the TSA, i.e., the AskForInformation() and AcceptInformation() operations. These need migration of the TBA to the TravelProvider AgentSystem but this migration is not considered in the behavioral specification.

- The local interaction SetRetreatedInformation() with the PTA to pass the information when it returns back to Organizer AgentSystem.

The designer supplements this behavioral specification by marking these migration points in the behavioral specification of the agent and by adding the elements related to the migration, namely the move() operation in the description of the AskForInformation() and AcceptInformation() operations. In order to specify which migration point corresponding to which current part of the behavior must be considered, the designer makes use of a variable. Depending on its value, one of the three parts of the behavior is performed. This variable is introduced in the beforeMove() and afterMove() methods specified by the designer when he/she wants a state migration. It is updated in these methods in order to consider the passage of a marked migration point, i.e. the beginning of another behavior part in an agent’s lifetime.

6 Conclusion

In order to be competitive, the telecommunication providers need to improve their services development techniques. Today, a major trend in software engineering is the model-oriented development, recognized as a factor of productivity in the software development. The benefits of using this approach in mobile agent-based telecommunication services are the rapid introduction of new services available for the end-users. Mobile agent-based services are built upon platforms that provide facilities for the agents’ execution. The MASIF standard is the OMG initiative to unify a set of principles for the platforms’ construction. It can be considered only as a first step for the standardization of concepts and operations that a mobile agent environment has to provide. It defines the entities of a mobile agent platform but it is more focused on the localization aspect of the interoperability and leaves the other distribution concerns be decided by the developers of mobile platforms. Nevertheless it provides a base for the description of the execution platform distributed aspects that need to be added to a behavioral specification. Our approach considers MASIF and
ODP as the overall standards in dealing with these distribution aspects [11]. Using such standards, we provide an ADL in the form of a UML profile, enabling a designer to specify his/her mobile agent-based service. This leads to an integrated system architecture specification capable of assisted code generation.

References

7. Miller J. and Wirfs-Brock R., How can subsystems be both a package and a Classifier, (www.omg.org)
8. FIPA, www.fipa.org
9. www.grasshopper.de
10. Geswin Thomas, Object-Oriented Mobile Agent Systems,, ECOOP workshop 2000, Springer-Verlag
Multi-domain Policy Based Management
Using Mobile Agents

Nazim Agoulmine\textsuperscript{1,2,3}, Mauro Fonseca\textsuperscript{4,5}, Alan Marshall\textsuperscript{6}

\textsuperscript{1} LIP6 Laboratory, University of Paris VI, France
\{Nazim\_Agoulmine, Mauro.Fonseca\}@rp.lip6.fr
\textsuperscript{2} LSC Laboratory, University of Evry, France
\textsuperscript{3} UQAM, University of Quebec at Montreal, Canada
\textsuperscript{4} Advanced Telecommunications Systems Laboratory,
School of Electrical & Electronic Engineering,
The Queen’s University of Belfast Northern Ireland, U.K.
A.Marshall@ee.qub.ac.uk
\textsuperscript{5} Pontifical University Catholic of Paraná, Brazil

Abstract. This paper describes a new framework for the interoperability be-
tween ISP management domain for the purpose of satisfying end user require-
ment based on service level agreements (SLA) set up between a customer and
its related ISP and also SLA set up between ISPs. The paper considers future
policy based enabled equipments and management centers based on the ongo-
ing work undertaken in the frame of the resource allocation protocol and policy
framework groups of the IETF. The objective of this paper is to investigate the
possibility to merge policy based management with mobile agents in order to
handel QoS of communications spanning over a number of ISP domains. In
this environment, mobile agents will act on behalf of users or third party ser-
vice providers, to obtain the best end to end service based on a negotiation
process between ISP policy management systems.

1 Introduction

Policy based management is a gaining approach to deploy management strategies.
In the context of the Internet, the complexity of the composition of the Internet neces-
sitates a close negotiation between ISP's (Internet Service Provider) in order to pro-
vide value added connectivity services. In the POTS ( Plain Old Telephone Service)
network, these agreements were achieved between telecommunication operators for
the purpose to establish an international phone call service with guaranteed QoS. In
the Internet, the number of services can be enormous and it is difficult to achieve a
global agreement on the overall services. Thus ISP can negotiate cooperation on ser-
vice per service base. The set of agreement ISP will agree on will be defined in ISP-
to-ISP SLA. These agreements are the formal negotiated agreement between an ISP
Provider and an ISP Customer for service delivery. It is designed to create a common
understanding about services, priorities, responsibilities, etc. SLAs can cover many
aspects of the relationship between the ISPs such as quality of services, customer care, billing, provisioning etc. Similarly, end users connected to a particular ISP have agreed with this latter for Customer-to-ISP SLA.

When the service requested by a customer span a number of ISP, negotiation between ISP has to take place in order to assume to the customer the best deal for its request. For instance, if the ISP has connectivity with two other ISPs, there should be a process that allows searching for the best service (for instance, in term of QoS or Price) on a customer-based requirement defined by the SLA.

In this paper, we suppose that ISP will deploy in the near future, policy based management systems. Policy defines a set of rules that govern the behavior of the network depending on SLA. The purpose of this paper is to investigate the possibility to facilitate the negotiation between ISP for the purpose of satisfying a customer request. Each ISP will establish SLA with a customer and with other ISPs. When a customer apply for a service, it is necessary to set up a process that will permit to verify if the service can be assumed depending on various parameters such as, the customer, the type of service, the date/time, etc. The developed framework proposes to use mobile agents to facilitate the implementation.

The remainder of the paper is organized as follows: section 2 describes the background concepts for the purpose of this work. Section 3 presents the objectives of this work. The forth section presents the proposed framework for interdomain policy based management using mobile agents. Section 5 describes the architectures of the different components of the framework. And finally a conclusion and future works.

2 Background

In this work we have addressed a number of concepts: policy based management, agent technology; common information model which are introduced briefly in this section.

2.1 Policy Based Management

The policy based management [1] approach aims to defines high level objectives of network and system management based on a set of policies that can be enforced in the network. Policies are a set of pre-defined rules (defined actions to be triggered when a set of conditions are fulfilled) that govern network resources, including conditions and actions that are established by the network administrator with parameters that determine when the policies are to be implemented in the network. In the case of ISP, policies are defined based on one hand the high-level business objectives of the ISP and on the other hand on the SLA (Service Level Agreement) agreed with its customers and partners ISP. The Policy Working Group [2] of the Internet Engineering Task Force is chartered to define a scalable and secure framework for policy definition and administration [3][4]. The main goal is to support QoS management. This group has defined a framework for policy based management that defines a set of component to
enable policy rules definition, saving and enforcing. It identifies two primary main components by their functionality. The framework is comprised of a Policy Enforcement Point (PEP) that is a policy decision enforcer component and a Policy Decision Point (PDP) which is the decision-making component.

2.2 Agent Technologies

The agent concept has been widely proposed and adopted within both the telecommunications and Internet communities as a key tool in the creation of an open, heterogeneous and programmable network environment [5]. This trend is motivated by the desire to use the agents to solve some of the problems encountered in large scale distributed and real-time systems such as the volume and complexity of the tasks, latency, delays, and others. Generally, an agent can be regarded as an assistant or helper, which performs routine and complicated tasks on the user's behalf. In the context of distributed computing, an agent is an autonomous software component that acts asynchronously on the user's behalf. Agent types can be broadly categorized as static or mobile [6],[7]. The main motivation of the use of agent technology in this work is driven by the desire to automate the control and management processes by allowing for more programmability of the network to rapidly customize the provision of new information and telecommunication services [8],[9],[10].

2.3 Common Information Model,

The work undertaken by DMTF (Desktop Management Task Force) for the purpose of integrated system and network management has leaded to the definition of a common information model called CIM [11]. CIM is an implementation neutral schema for describing overall management information. It has been adopted by IETF, aims to establish a common conceptual information model that captures every notion that is applicable to all areas of management including policy definition. This model is extended in this work in order to support the modeling of network and policy information as well as service level agreements.

3 Objective of this work

The ongoing panorama of networking shows a numerous number of ISP located at different geographical area. At the same time, companies are requesting more sophisticated services that permit them to connect between their sites or with sites of other companies in a personalized way.

In the context of fierce competition in open liberalized telecommunications markets, network providers are therefore currently investigating opportunities to provide their customers with differentiated service level agreements (SLAs) which state the obligations entered into by both network provider and customer
However, to satisfy the customer needs, it is mandatory to take its requirements into account in a flexible way even if the management of the end to end communication is more challenging since the service can span heterogeneous network provider domains. And yet needs to be managed on an end-to-end basis.

Thus it is necessary to enhance the PBM framework in order to take into account the multi-party process of policy based management. In fact, ISP establishes a set of agreement with others ISP in order to provide an end to end service to customer. Agreement between ISP will be based on ISP to ISP SLA that can change during time according to the business strategies. However, it is necessary to automate the interaction process between ISP Policy Based Management in order to hide the complexity of the end to end management. Interdomain PBM have to provide facilities to adapt quickly to new changing strategy regardless the relation between a particular ISP and the other ISP. For instance, ISP can have different agreement with other ISP to provide connectivity to the same destination.

In this paper we investigate the possibility to use mobile agents as flexible approach to PBM over multi-domain IP networks. We suppose that each provider has deployed a policy-based management in its own domain. Each provider has its own business objectives that can changes rapidly depending on the economic context, the economic strategy followed by the operator and agreements between the various network providers at a wide area scale.

4 Proposed framework

Because of the complexity of the policy management process in the context of multi-domain operators and its implementation and security issues, it is likely that the client/server policy based management approach will be replaced by a mobile agent approach. We call this management architecture “Mobile Agent PBM ” (MA-PBM). It can avoid scalability problems and offers flexibility to users, third party operators and network operators as it will be shown in the following sections.

The open framework defines a set of agents depending on their respective roles in the architecture:

<table>
<thead>
<tr>
<th>Type</th>
<th>Role</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local PEP Agent Manager:</td>
<td>It is a fixed agent that performs local routine control/management PEP functions. It performs mainly metering and enforcement functions as well as the creation/deletion of PEP mobile agent when it needs to interact with the PDP for decisions.</td>
<td>Local agent, no mobility.</td>
</tr>
<tr>
<td>PEP Mobile Agents</td>
<td>These are used mainly as autonomous negotiator agents between PEP and PDP within the same domain. The PEP Mobile agents are used to obtain decisions from PDP. The PEP mobile agent carries all the information regarding the ongoing connection. It is sent by the PEP to the PDP in order to notify a particular event in the Intra domain agent, mobility capabilities.</td>
<td></td>
</tr>
</tbody>
</table>
network (RSVP opening request [12], QoS degradation, etc).

| **Local PDP Agent Manager** | It is a fixed agent that takes decision regarding the information that is carried out by the PEP Mobile Agents. It interacts with the various databases (policy rules DB, MIB, security DB, etc) in order to retrieve the rules that can be triggered. Once the decisions are identified, it gives them to the PEP mobile agent. If any configuration related to new policies are defined, it creates a PDP Mobile Agent, it send to remote PEP to perform the new configuration. It takes also into account inter-domain interactions, when a decision needs to be negotiated with remote domain PDP Agent Manager. When interacting with remote domain, it creates a PDP Domain Mobile Agent. | Local agent, no mobility. |
| **PDP Mobile Agent** | When a PDP has taken a decision it sends a PDP Mobile Agent to enforce policies directly in the PEP component in all the network elements that are concerned by this new decision. | Intra domain agent, mobility capabilities. |
| **PDP Domain Mobile Agent** | When a PDP has to take a decision related to an inter-domain connection, it has to identify the set of remote domain need for the connection and send a Domain Mobile Agent to negotiate the term of services needed by the customer. | Inter domain agent, mobility capabilities. |

4.1 Domain Interaction for a service spanning two ISP domains

In the case of two ISP domains interconnecting the customers premise networks, the deployment of the different agents in the global distributed architecture is described in the following figure:

![Figure 1. Architecture of Interdomain PBM](image-url)
The local PDP agent is responsible for collecting information related to the entire domain. If any change occurs in the network such as an RSVP connection request, the local PEP agent running on the ingress router creates a PEP mobile agent and sends it to the PDP system. The sent agent contains all the information needed to identify the source of the request (customer) and the destination of the call (calling party) as well the parameters related to this event (for example QoS parameters for the request RSVP connection). Based on this information, the PDP local agent retrieve related information and policies from the policy DB, the MIB and the security server using different types of protocols such as LDAP, SNMP or any other protocol that permit to retrieve information from a database. Then, the local PDP agent tries to trigger any policy rule that can be triggered regarding the information carried by the PEP mobile agent. If the connection doesn't span a different ISP domain, the PEP mobile agent carries back the response to the PEP local agent. If the decision needs to interact with remote ISP, the local PDP agent, sends a PDP domain Mobile agent to remote ISPs with all information related to the requested service as well as information permitting to identify the initiating domain. The remote PDP local agent gets the necessary information from the remote PDP domain mobile agent. According to this information, it tries to trigger any policy rule that defines the ISP-to-ISP policy rules between this ISP and the initiating ISP defined within the SLA. If the service is accepted the PDP domain mobile agent, collect all the information related to the decision and move back to its domain. The local PDP agent retrieves the information and takes a final decision regarding the request service.

When the final decision is taken, each local PDP agent of each domain that intervene in the final decision has to configure its own equipment's in order to enable the customer service to be operational. This means for instance to enforce policy directly into equipment's using PDP mobile agents. Consequently PDP mobile agent will move from one equipment to another in order to enforce locally the policy by interacting with the local PEP agent.

4.2 Domain Interaction for a service spanning three ISP domains:

The described process can be complex in the case of numerous ISP that interconnect the two remote sites with different agreements. Thus negotiations have to be set up with different ISP in order to found out best solution according to different criteria’s such as pricing, QoS, duration and so on. In fact, price for example, can vary according to a network operator’s tariffing policy, and according to the competition between different operators.

In case of three ISP domains as described in the figure 2, the PDP inter domain agent will move from one domain to another in order to interact with the local PDP agent manager. As in the previous example, the PDP inter domain will carry all the necessary information to trade with the remote local PDP agent for the purpose to obtaining a response for the requested service. If one of the remote local PDP agent refuse to serve the PDP inter domain agent regarding its local management policies, the PDP inter domain agent move back to it initial domain and inform its initiator.
agent of the negative response. However, in case of success, the PDP interdomain agent continues its trip until the latest domain. During the travel, the agent obtains authorization to move to a different domain for the purpose of trading the end to end service.

In case of acceptance of the end to end service, the interdomain mobile agent informs each PDP local agent in the way back to the initial domain of the final decision and collects and distributes the SAP necessary for the service initiation between ISP domains. As a matter of fact, each PDP local agent creates a PDP mobile agent and sends it towards the various routers for local configuration by the PEP local agent as described in the figure 2.

5 System architecture and information model

The system architecture comprises two main components, the PEP and the PDP environments as described below. These two environments differ mainly in term of localization. In fact the PEP environment is located at the router boundary while the PDP environment is a stand-alone system. The PEP environment has to be very light in the sense that it should not require a lot processing and memory resources and should be as faster as possible.
5.1 Policy Enforcement Point Architecture

The execution environment at the PEP point is a mobile agent agency. The agency is a MASIF like middleware located in the router. In this scheme, a MASIF like middleware based on a JVM is proposed as an technological architecture for agent execution. Based on the PDP decision, the local PEP agent assigns a policy to users' connection. In order to have a standard interface between the agents deployed on the router and the embedded hardware and software, an ORB (Object Request Broker) is used between the JVM and the Kernel. The reason for using an ORB is to provide in one hand all the support for agent management and mobility and on the other hand a standard L interface [13] to interact with router kernel, since there is a wide variety of hardware and software within routers from different vendors.

5.2 Policy Decision Point Architecture

The execution environment at the PDP point is also a mobile agent agency. The agency is based also on a like middleware located in a stand-alone system. This environment should provide facilities for policy rule directory access, security server access and MIB access. The access to the policy rule directory is performed using LDAP (Lightweight Directory Access Protocol). Access to security server can be performed using telnet or any other useful protocol. The MIB access is realized using the SNMP (Simple Network Management Protocol) as it is a standard for such access.

5.3 Interdomain Policy Management Information Model

The information model specified to capture interdomains interaction functionality is derived from the DMTF CIM (Common Information Model)[11]. For simplification reasons, the classes not used for this model are not described in the figure 4. Mainly, the information model for a particular PDP environment permits to capture the information related to the customer connected to the ISP as well as the information related to the remote ISP which a contractual relationship with this particular ISP as described in the following figure:
The simplified model presents a routing table that permits to identify the route to be used for negotiation. In case of different routes for the same destination the PEP local agent creates a PEP interdomain mobile agent of each route. Each created agent will be sent in one direction with the requested parameters for the route in order to trade with the remote PDP local agent for the purpose of a customer service deployment. The routing inside a particular domain was not considered as far as we consider that it exists a local domain routing protocol that are able to identify the route inside the domain to satisfy the customer requirements.

7 Conclusion and future work

In this paper we have presented an integrated framework for interdomain policy based management based on contractual relationships between the customers and an ISP on one hand and between ISPs on another hand. These contractual relationships are described in term of policy rules in each domain. The policy defines the set actions to perform when particular events occur. The idea is to define a flexible and efficient solutions for a problem of service deployment over different Internet domains. Existing approach to offer end-to-end QoS are static and makes difficult to set up in the physical network. Hence it is not possible to react quickly to customer changes. Policy based management framework offers a good starting point to automate the process in one domain, however the issue of interdomain policy based management is still open. The proposed approach uses a set of agent with different skills. Each ISP is responsible for its domains and can change its business strategies without changing anything in the system. The interaction process between domain is performed automatically and any changes in the policies are taken into account when a
service has to be set up in the particular domain. Hence, the specified framework
considers a number of key technologies to deploy the overall system. The technolo-
gies employed include mobile agent platforms, MASIF, CORBA[14]. The framework
also identifies different levels for the implementation of these technologies within the
network.

Many aspects of this work are not completely resolved. It is a first attempt to ad-
dress policy based management in multidomain using mobile agent. The following is
to go deeper in the specification of the agent interactions as well as the information
model according to the recent progress in the IETF policy group.

8. References

   (www.ietf.org)
   charter.html.
   (www.qosforum.com)
   Intelligent Network Implementation”, IEEE Network Magazine, “Special Issue on Active
6. H. S. Nwana and M. Wooldridge, «Software Agent Technologies». BT Technology Journal,
7. David Chieng et al., “A Mobile Agent Brokering Environment for The Future Open Net-
   work Marketplace”, Seventh International Conference On Intelligence in Services and Net-
8. A.Marshall, N.Agoulmine, "An Agent-based paradigm for managing service quality in open
   network environments", published in the International Journal on Network and Service
   Management, Wiley publisher.
10. N. Agoulmine, D. Dragan, T. Gringel, J. Hall, E. Rosa, M. Tschichholz, "Trouble Manage-
    ment for Multimedia Services in Multi-Provider Environments", International Journal on
11. Systems Management: Common Information Model (CIM), Open Group Technical Standard
    C804, DMTF. ISBN 1-85912-255-8, August 1998..
    (RSVP) -- Version 1 Functional Specification”, RFC 2205, Network Working Group, IETF,
    September 1997. (www.ietf.org)
    pared by the Working Group for IEEE 1520 on Application Programming Interface for
    Networks), http://www.iss.nus.sg/IEEEPI.
14. OMG Mobile Agent System Interoperability Facility (MASIF) Specification,
A Mobile Agent Prototype
for Distributed Search

Elio Velazquez, Nicola Santoro, and Amiya Nayak

Center for Parallel & Distributed Computing
School of Computer Science, Carleton University, Ottawa
Ontario, Canada, K1S 5B6
{evelazqu, santoro, nayak}@scs.carleton.ca

Abstract. In this paper, we propose a mobile agent platform as an alternative to a message passing approach to solve a distributed search problem. The search problem is analyzed from two different perspectives, the single travelling agent case and the multiple travelling agents case. We conduct several experiments to measure the performance of the mobile agent solutions as well as the stationary agents for different network topologies and observe that the multiple agents approach performs better when the number of nodes in the network increases. The tree topology, in particular, gives the best overall performance among all topologies considered.

1 Introduction

Mobile agent computing, considered as a special case of message passing, attempts to move computations as close as possible to the data and makes efficient use of the bandwidth by considerably decreasing the number of messages exchanged between cooperating applications (see [1], [3], [5], [6], [11]). Agents are simply programs which help accomplish a task without a continued interaction with the user. They can be pro-active or reactive and more importantly, stationary or mobile. Mobile agents by themselves cannot exist in a networked environment without a platform that guarantees the most important and distinctive property: migration. Mobile agent systems or mobile agent platforms offer an environment for agent execution. They are responsible for executing the agent, sending the agent across the network and reactivating incoming agents.

In this paper, we propose a model to solve a generic search problem based on mobile agent technology. The architecture, we propose, involves two major approaches, mobile agents and stationary agents. It consists of several components to guarantee the agent coordination and interactions as well as some fault tolerance mechanisms. We reduce a distributed search problem in a network of arbitrary topology to the implementation of a traversal algorithm applied to a logical spanning tree constructed on top of the original topology. Solutions for the single and multiple travelling agents are provided along with supporting components to implement agent coordination. Using TACOMA mobile agent platform [7], we develop a prototype, which explores several variables that may
affect the performance of a mobile agent solution, such as topology, size of the network and number of agents involved in the search. The test results show that the number of agents has a positive direct impact on performance when the size of the network increases. Furthermore, the network topology does impact the overall performance. For these particular experiments, the binary spanning tree shows the best performance among other topologies tested.

2 Proposed Mobile Agent Architecture

The proposed model comprises several components, which are responsible for the exchange and coordination of cooperating agents. Along with the mobile agent platform, our architecture defines mobile and stationary agents. Fig. 1 shows the architecture of the proposed model along with all the components involved. Each node of the network contains the mobile agent platform and stationary agents such as: whiteboards, routers and blackboards. The routers are implemented as stationary agents. Mobile searcher agents can meet either locally or remotely with these routers and obtain information about any node of the network. Agents perform a dummy search at each site consisting basically of opening a file a reading the content.

In our scheme, mobile agents request information about the local neighbours when they arrive to a site. This information is provided via a stationary agent called an agent router. This interaction can be seen as a particular case of meeting interaction. Every node of the network has a stationary agent router, which provides information about local neighbours and may provide information about neighbours of any other node of the network. This stationary agent acts as a local or global router to facilitate the navigation of mobile agents.

The implementation of the agent router is based on the router pattern [4]. The objective of this pattern is to solve how agents can select a destination where a task can be performed best. The agent router sorts the list of local neighbours based on an estimation of the time required to perform the task at each node.

![System architecture](image)

Fig. 1. System architecture
Likewise, the implementation of the whiteboard is based on the whiteboard pattern [4]. This pattern solves the problem of how agents can exchange location specific data with other mobile agents. The whiteboards are implemented as TACOMA cabinets. These cabinets are persistent structures where travelling agents can register upon arriving at a site. This structure allows an agent to leave information in the form of TACOMA’s folders to be read by incoming agents. TACOMA provides an API to create and manage this kind of structures. The blackboards, on the other hand, are implemented as independent TCP/IP multithreaded servers that agents can contact to inform about the creation of new agents and to inform about the completion of a partial search. The blackboard allows agents to register every time they duplicate themselves. The blackboard, if requested, informs the number of agents registered at a given time. There exists one blackboard per node and agents will report to the initiator’s blackboard. The coordination model implemented combine direct coordination and whiteboard coordination techniques ([2]).

3 Agent and Message Based Solutions for Distributed Search

In this section, we present the distributed search algorithms from the message passing and mobile agent perspectives. In both approaches, the single travelling agent/message and the flooding technique are described separately. Both solutions, messages and mobile agents, consist of two major parts: the launching application and the travelling message or agent. The launcher application is responsible for packaging the agent into a TACOMA briefcase and injects it into the network. Once the agent or message has been sent out, the launcher or initiator blocks until the search is complete.

The agents are implemented as a very simple finite state machine, with three states: INIT, SEARCHING and DONE. Initially, when the initiator creates an agent, its state is set to INIT. When the agent reaches the first searchable node in its itinerary, its state is changed to SEARCHING. As the agent or message moves across the network, it remains in the SEARCHING state. Only when the HOST folder is exhausted, or in other words, when the agent has fulfilled its itinerary, its state is changed to DONE. Agents or messages in DONE state have to travel to the initiator to report their partial search results.

The interactions between agents and the stationary components responsible for the coordination are represented in the algorithms by employing supporting functions. In particular, the interactions between the mobile agents and the agent router are encapsulated in the functions Get_Local_Neighbours and Get_Global_Neighbours. The Get_Local_Neighbours function performs a local meeting with the agent router in order to obtain the list of immediate neighbours. On the other hand, the function Get_Global_Neighbours is able to contact either a local or remote agent router and make inquiries about immediate neighbours of any node in the network.
The direct coordination mechanism can be found in the backboard related functions: \texttt{Insert\_AgentID\_in\_Blackboard} and \texttt{Remove\_AgentID\_from\_Blackboard}. Both functions establish a TCP/IP connection with the initiator’s blackboard in order to set and retrieve agent related information. Similarly, the whiteboard-based coordination is implemented by the functions \texttt{Get\_AgentID\_From\_Whiteboard} and \texttt{Set\_AgentID\_In\_Whiteboard}.

### 3.1 The Agent Based Algorithms

We now present the algorithms for the single travelling agent and the flooding scheme. These algorithms describe the behaviour of the mobile agents after being injected into the network by the initiator or launching application. The algorithm starts with the mobile agent in the INIT state. This means that the agent was successfully created and is ready to interact with the local resources of the first searchable node in its itinerary.

Every time the agent arrives at a node other than the initiator, it performs a local search and carries the result in a special folder called DATA. If the agent reaches the initiator and its state is set to DONE. The agent then contacts the launching application to inform the partial search results. After performing a local search, the agent retrieves the list of local neighbours. To achieve this, the local agent router is contacted by calling the function \texttt{Get\_Local\_Neighbours}. The list of neighbours obtained is inserted in the HOST folder. This is the folder that the agent carries along that specifies the itinerary. Different implementation of the \texttt{Insert\_Local\_Neighbours} function may produce different itineraries.

If the list of neighbours returned by the local router is empty, it means that the node does not contain any immediate neighbour. In this case, the agent sets its state to DONE and returns to the initiator. If the list of neighbours is not empty, the agent attempts to travel to the first node in the list. If the agent cannot complete the migration because the remote node is temporarily down or due to any communication error, it contacts the global router. When a global router is contacted, it informs the list of immediate neighbours of that faulty node.

#### The Single Travelling Agent

If \texttt{INITITOR} and \texttt{STATE=DONE} 
Begin 
\texttt{Inform\_Partial\_Results} 
\texttt{Exit} 
End 
Else 
\texttt{STATE=SEARCHING} 
\texttt{DATA = Perform\_Local\_Search} 
\texttt{Get\_Local\_Neighbours (ROUTER)} 
\texttt{Insert\_Local\_Neighbours (HOST\_LIST)}


For each Ni in HOST_LIST
Begin
    If Ni == NULL // Ni is the last host in list
        Begin
            STATE=DONE
            Travel to INITIATOR
            Exit
        End
    Travel to Ni
    If cannot_travel to Ni
        Begin
            Get_Global_Neighbours (Ni, ROUTER)
            Insert_Local_Neighbours(HOST_LIST)
        End
    Else Exit
End

Flooding Algorithm for Mobile Agents

If INITIATOR and STATE=DONE
Begin
    Inform_Partial_Results
    Remove_AgentID_from_Blackboard
    If Blackboard Empty
        Notify_Launcher_Application
    Exit
End
Else
    STATE=SEARCHING
Get_AgentID_From_WhiteBoard
IF AgentID // this host has been already visited
    Travel to INITIATOR
Else
    Set_AgentID_In_Whiteboard
Perform_Local_Search
Get_Local_Neighbours (ROUTER)
Insert_Local_Neighbours (HOST_LIST)
For each Ni in HOST_LIST
Begin
    If Ni = NULL // Ni is the last host in list
        Begin
            STATE=DONE
            Travel to INITIATOR
            Exit
        End
End
Travel to Ni
If cannot_travel to Ni
Begin
   Get_Global_Neighbours (Ni, ROUTER)
   Insert_Local_Neighbours (HOSTLIST)
End
Else
   Insert_AgentID_in_Blackboard
End

3.2 The Message Passing Algorithms

Similar to the mobile agent approach, the message passing solutions utilize the TACOMA system as a communication platform. In this particular case, the agents are stationary. The communication is limited to the exchange of a briefcase, which contains data and additional information needed to coordinate agent actions, such as the global router and the initiator. The stationary agents exchange messages according to the same techniques employed in the mobile agent case. This allows us to compare the solutions within the same group (agents or messages) or between the two approaches.

The message passing algorithms are similar to their agent based counterparts with only difference is that there is no agent migration in the message passing case, so a message is passed instead. Therefore, the message passing algorithms are omitted.

4 Experimental Results

All the experiments were carried out in the Graduate Lab of the School of Computer Science in the absence of simulated failures and under normal circumstances. The computers were used simultaneously with other users and applications and no special care was taken to guarantee an exclusive access to computer resources or network during these experiments. Since all the tests were carried out in an actual networked environment, the limitations on the number of machines involved in the experiments were determined by the number of nodes available at the Graduate Lab of the School of Computer Science.

The prototypes were tested to measure the performance of the different approaches with the objective to obtain sufficient information to compare the performance between the mobile and stationary agents as well as the performance within the same category (agents or messages). In the experiments, agents were injected from any node of a network with the purpose of performing a distributed search. The launching application or initiator measured the time in milliseconds (calling the function ftime) before injecting the first agent and after the last
agent had arrived. The difference between these two measurements was considered as the execution time for the entire search. Figs. 2 and 3 show the average execution times for the single and multiple travelling agent as well as the single travelling message and the flooding of messages. A complete listing of the execution times for each experiment can be found in [10].

The message passing solutions performed much better than the mobile agent solutions. For the message passing approach, the agents are stationary, and they only exchange a briefcase containing the partial search results (information gathered at each node) and other information necessary for coordination. Although not as clear in the message passing approach as in the mobile agents, the multiple agents scheme appears to be the most efficient mechanism to implement in a distributed search on the spanning tree.

In the next set of tests, we considered the single travelling agent implementation and the flooding implementation in several networks with different sizes.
Fig. 4. Average execution times for a single travelling agent

topologies, keeping the network size constant at 16 nodes (N=16). The topologies compared were ring (with bi-directional and unidirectional links), hypercube of order 4 (16 nodes) and a binary tree. The Figs. 4-7 show the average execution times of the two solutions (single travelling agent and the flooding) applied to these topologies.

For the case of a single travelling agent, the execution times were similar for the tree and the ring topologies. This is the case for a single travelling agent as well as for a single travelling message but not for the hypercube, which showed the worst execution times for both algorithms in the two approaches. The hypercube also had the worst execution times in the multiple travelling agents or flooding scheme. In this case, the bi-directional ring performed better as expected, thanks to the parallelism as there were two agents travelling in opposite directions. For the hypercube, even though there is parallelism as well, it seems

Fig. 5. Average execution times for multiple travelling agents
that the overhead required to clone agents and verification of multiple visits as well as communication with the blackboard, constitutes an extra overhead eliminating any additional advantage that parallelism may offer.

Unlike other experimental results ([1], [8], [9]) where under certain circumstances the mobile agent solutions outperformed the message passing implementations, the stationary agents performed better than the mobile agents for all cases in our experimentation. The multiple agents or messages scheme performed better than the single travelling agent or message with the increase in the network size. Multiple agents introduce a desirable level of parallelism in the search that for larger number of nodes seems to counteract any negative impact produced by the overhead of coordinating the agents.

Our results also show that the topology of the network does have an impact on the overall performance. In this case, the tree topology has shown the best
performance among the topologies considered. In particular, the binary tree has the best response times when compared with other trees with larger number of neighbours per node.

References

Using the Cross-Entropy Method to Guide/Govern Mobile Agent´s Path Finding in Networks

Bjarne E. Helvik and Otto Wittner
Department of Telematics, Norwegian University of Science and Technology
{bjarne, wittner}@item.ntnu.no

Abstract. The problem of finding paths in networks is general and many faceted with a wide range of engineering applications in communication networks. Finding the optimal path or combination of paths usually leads to NP-hard combinatorial optimization problems. A recent and promising method, the cross-entropy method proposed by Rubinstein, manages to produce optimal solutions to such problems in polynomial time. However this algorithm is centralized and batch oriented. In this paper we show how the cross-entropy method can be reformulated to govern the behaviour of multiple mobile agents which act independently and asynchronously of each other. The new algorithm is evaluate on a set of well known Travelling Salesman Problems. A simulator, based on the Network Simulator package, has been implemented which provide realistic simulation environments. Results show good performance and stable convergence towards near optimal solution of the problems tested.

1 Introduction

The problem of finding paths in networks is general and many faceted with a wide range of engineering applications in communication networks. Examples: end to end paths in (virtual) circuit switched networks both for primary paths and backup path in SDH, ATM and MPLS, routes in connectionless networks, shortest (or longest) tours visiting all nodes (STST). Path is used as a collective term encompassing a number of the more specific technical terms path, route, circuit, tour and trajectory.

Finding the optimal path or combination of paths usually leads to NP-hard combinatorial optimization problems, see for instance [1, 2]. A number of well known methods exist for solving these problems, e.g. simulated annealing, [5], tabu search [3] genetic algorithms [4] and the Ant Colony System [9]. A recent and promising method, the cross-entropy method, is proposed by Rubinstein which finds a near optimal solution in polynomial time (O(3)) [6]. However, when we implement path finding as a management functionality of a network, we have another additional requirement, which is not easily met by the above algorithms:

- The algorithm should be distributed, i.e. the path should be decided by a cooperative task among the network elements. This increase the dependability of the network by avoiding the single point of failure of a centralized network management system and avoid that the management rely on the network that is managed.

Multiple mobile agents, exhibiting an insect like swarm intelligence, has been proposes as a means to path finding in communication networks in a distributed and adaptive manner [10, 11, 12, 13]. Hereto, these mobile agent systems have concentrated on solving the shortest path routing problem. A more general approach is desirable to enable
implementation of a wider range of management applications. Constructing systems capable of finding good solutions to the travelling salesman problem (TSP) may fulfill this generality since TSPs are among the hardest routing problems (NP complete).

In this paper we will show how the cross-entropy method of [6], which has been evaluated successfully on TSPs, can be reformulated to govern the behaviour of multiple mobile agents towards finding optimal paths in networks. This reformulation is presented in Section 4. How this behaviour is implemented in the Network Simulator [8] is presented in Section 5. The ability to find (near) optimal paths are demonstrated through some case studies in Section 6, before we conclude. First, however, an introduction to path finding by multiple agents is given in Section 2 and a brief review to the cross-entropy method in Section 3.

2 Path Finding by Multiple Agents

Schoonderwoerd & al.’s paper [10] introduces the concept of multiple mobile agents cooperatively solving routing problems in telecommunication networks. A number of simple agents move themselves from node to node in a network searching for paths between a given pair of source and destination nodes. A probability matrix, represented as probability vectors in each node, controls the navigational behaviour of the agents. When a path is found the probability matrix is adjusted according to the quality of the path such that a better path will generally have a higher probability of being reused. By iterating this search process high quality paths emerge as high probabilities in the matrix.

We regard a network with \( n \) nodes with an arbitrary topology, where the only requirement is that it is feasible to establish the required path. A link connecting two adjacent nodes \( k, l \) has a link cost \( L_{kl} \). The link cost may be in terms of incurred delay by using the path, “fee” paid the operator of the link, a penalty for using a scarce resource like free capacity, etc. or a combination of such measures.

Path \( i \) through the network is represented by \( \pi_i = \{r_1, r_2, ..., r_{n_i}\} \) where \( n_i \) is the number of nodes traversed. For a TSP tour \( n_i = n + 1, \forall i \) and \( r_1 = r_{n_i} \).

The cost function, \( L \), of a path is additive,

\[
L(\pi_i) = \sum_{j=1}^{n_i-1} L_{r_j r_{j+1}}.
\]  

(2.1)

The foraging behaviour of ants has so far been the major inspiration for all research on multi mobile agent systems for routing. When an ant has found a food source it marks the route between its ant hill and a food sources with a pheromone trail. Other ants searching for food will with a higher probability follow such a trail than move about randomly. On their way home from the food source they will reinforce the pheromone trail and increase the probability of new ants following the trail.

Viewing mobile agents as artificial ants and network nodes as the environment we can interpret pheromone trails as routing probabilities. We have an unconditioned probability \( p_{t,rs} \) of an agent choosing to go to node \( s \) when it is in node \( r \) at time \( t \). The actual
choice of next node may be conditioned on the agents past history according the selection strategies of the agents (Section 4.2). We denote the set of unconditional routing probabilities as \( p_t = \{ p_{t,r,s} \} \). The probability of choosing a specific path, \( \pi \), under the current selection strategies is \( p_t(\pi) \) which is uniquely determined by \( p_t \).

### 3. The Cross-Entropy Method

A new and fast method, called the cross-entropy method, for finding the optimal solution of combinatorial and continuous nonconvex optimization problems with convex bounded domains is introduced by Rubinstein [6]. To find the optimal solution, a sequence of simple auxiliary smooth optimization problems are solved based on Kullback-Leibler cross-entropy, importance sampling, Markov chain and Boltzmann distribution. In the rest of this section we review the method and state some results in the context of the problem at hand. For details it is referred to [6].

The basic notions of the method is that in a random search for an optimal path, the probability of observing it is a rare event. For instance, finding the shortest travelling salesman tour in a fully meshed network with 25 nodes and an uniformly distributed routing probability from one node to the next is \( \frac{25!}{10^{25}} \approx 10^{-25} \). Hence, the probability of observing the optimal path is increased by applying importance sampling techniques [18]. However, doing this in a single step is not feasible. A performance function of the current routing probabilities, \( h(p, \gamma) \), is introduced:

\[
  h(p, \gamma) = E_p(H(\gamma, \pi))
\]

which is based on the Boltzmann function:

\[
  H(\gamma, \pi) = \exp\left(-\frac{L(\pi)}{\gamma}\right)
\]

*Fig. 3.1 Illustration of the Boltzmann function*

In (3.2) \( L(\pi) \) is denoted the potential function and \( \gamma \) the control parameter or temperature. It is seen that as the temperature decreases an increasing weight is put on the smaller path costs, see Fig. 3.1.

A temperature is determined which puts a certain emphasis on the shorter routes, i.e. the minimum temperature, \( \gamma_t \), which yields a sufficiently low performance function.

\[
  \min \gamma_t \quad \text{s.t.} \quad h(p^*_{t-1}, \gamma_t) \geq \rho
\]

where \( 10^{-6} \leq \rho \leq 10^{-2} \) and \( p^*_{t-1} \) is the current routing probabilities. The index \( t \) indicates the step in the iteration procedure and the initial routing probabilities \( p^*_0 \) is chosen to be uniformly distributed.
It is shown, [6], that the set of routing probabilities \( p^*_t \) which is the solution to

\[
\max_{p_t} E_{p_{t-1}} \left( \prod_{r_i \in \pi} H_{r_i r_{i+1}}(\gamma) \sum_{r_j \in \pi} \ln p_t, r_{j+1} \right) 
\]  

(3.4)

will minimize the cross entropy between the previous routing probabilities, \( p^{*}_{t-1} \), weighted with the performance function and \( p^*_t \), and represent optimal shift in the routing probabilities towards estimating the performance function with temperature \( \gamma_t \). In the above it is used that the path cost is an additive function which enables the Boltzmann function to be rewritten as

\[
H(\gamma, \pi) = \exp\left(-\frac{L(\pi)}{\gamma}\right) = \prod_{r_i \in \pi} \exp\left(-\frac{L_{r_{i+1}}}{\gamma}\right) = \prod_{r_i \in \pi} H_{r_{i+1}}(\gamma) 
\]  

(3.5)

It is shown that the solution to (3.4) is

\[
p^*_t, rs = \frac{\sum_{\forall i: \{r, s\} \in \pi} \prod_{j} H_{r_{j+1}}(\gamma) p^{*}_{t-1, r_{j+1}} \prod_{i} H_{r_{i+1}}(\gamma) p^{*}_{t-1, r_{i+1}}}{\sum_{\forall i: \{r\} \in \pi} \prod_{j} H_{r_{j+1}}(\gamma) p^{*}_{t-1, r_{j+1}} \prod_{i} H_{r_{i+1}}(\gamma) p^{*}_{t-1, r_{i+1}}} 
\]  

(3.6)

An optimal shift of routing probabilities, \( p_t^* \) toward the lower cost paths is obtained. We may now increment the iterator, \( t \leftarrow t + 1 \), lower the temperature by employing (3.3) to shift the emphasis further toward the smaller costs and find an improved set of routing probabilities. Hence, an iterative procedure is obtained which yields a sequence of strictly decreasing temperatures, \( \gamma_1 > \gamma_2 > \ldots > \gamma_t > \ldots \) and a series of routing probabilities \( p_0^*, \ldots, p_t^*, \ldots \) which almost surely converge to the optimal solution [6], where

\[
p_{t \to \infty, rs} = \begin{cases} 
1 & \{rs\} \in \pi^*, L(\pi^*) = \min_{\forall \pi} L(\pi) \\
0 & \text{otherwise}
\end{cases}
\]

Note that the above outlined method employs a global random search procedure, which is different from the local search heuristics of other well known random search algorithms for global optimization like simulated annealing, tabu search and genetic algorithms.

The procedure outlined is by Rubinstein applied in a batch oriented manner, i.e. a sample of \( N \) paths, is drawn from \( p^*_t \). On this basis the temperature is determined by the stochastic counterpart of (3.3), i.e. \( N^{-1} \left( \sum_{j = 1}^{N} \prod_{r_{j+1}} H_{r_{j+1}}(\gamma) \right) \geq \rho \), and routing probabilities by the stochastic counterpart of (3.6), i.e.

\[
p^*_t, rs = \frac{\sum_{j = 1}^{N} I(\{r, s\} \in \pi) \prod_{j} H_{r_{j+1}}(\gamma) \prod_{i} H_{r_{i+1}}(\gamma)}{\sum_{j = 1}^{N} I(\{r\} \in \pi) \prod_{j} H_{r_{j+1}}(\gamma) \prod_{i} H_{r_{i+1}}(\gamma)}
\]

1. \( N \) is typically chosen in the order of \( 10 \cdot n \cdot m \) to \( n \cdot m \), where \( n \) is the number of nodes in the network and \( m \) is the average number of outgoing links per node.
where $I(\ldots)$ is the indicator function. Rubinstein reports that empirical studies suggest that the cross entropy method has polynomial, in the size of the problem running time, complexity, e.g. $O(3)$.

The above result is valid both for deterministic link costs and for stochastic link costs [7]. Hence, the cross-entropy method may be used to find optimal paths in networks were the link costs are random variables like queuing delays and unused capacity. The application to such networks (obviously) is at the cost of larger sample sizes and/or more iterations.

4 Mobile Agent Behaviour

Studying the cross entropy method, it is seen that it forms the basis for a distributed implementation in a network using multiple simple mobile agents. The destination node of the agents keep track of the temperature. The agents move through the network according to the routing probabilities and the path selection requirements/constraints. For each path followed the cost is accumulated, cf. (2.1), which reflects the quality of the path. When a certain number of such paths have been found the temperature and the routing probabilities are updated.

However, a batch oriented decision of new temperature and new routing probabilities based on the information collected by a large number of agents, e.g. several thousands, is contrary to the basic ideas of swarm intelligence and is unsuited since it delays the use of the collected information, incurs storing of a large number of agents midway in their life cycle and a load peak when a probability update takes place. It also hampers the cooperation between families of agents. An incremental update of temperature and path probabilities is required.

4.1 Autoregressive Distributed Computations

To meet the requirement of an incremental update of temperature and routing probabilities, we have introduced autoregressive stochastic counterparts of (3.3) and (3.6).

When agent reaches its destination node the autoregressive performance function, $h_t$ is updated as

$$h_0 = \beta \cdot h_{t-1} + (1 - \beta) \cdot H(\gamma_0, \pi_0)$$

(4.1)

where, for the sake of notational simplicity the last arriving agent is indexed 0, second last $-1$, etc., and $\beta \in [0, 1]$ is the autoregressive memory factor, typically close to one.

In (4.1) the temperature after the agent has arrived is used immediately. If we had $M$ previously arriving agents, replace $h(p^i_{t-1})$ by $h_0$ in (3.3) and use (3.2), (3.3) may be rewritten as

$$\min \gamma_t \quad \text{s.t.} \quad \frac{1 - \beta}{1 - \beta^M + 1} \sum_{i = -M}^0 \beta^{-i} \exp(-\frac{L(\pi_i)}{\gamma_0}) \geq \rho$$

(4.2)
It is seen that the minimum is at equality. The equation is unsuited for solving in a network node since it is transcendental and the storage of an potentially infinite number of path costs \( L(\pi_j) \) is required. Assuming that the inverse of temperature does not change radically, a first order Taylor expansion of each term in (4.2) around the inverse of the temperature which were current when the corresponding agent arrived, is carried out, i.e.

\[
\rho \frac{1 - \beta^{M+1}}{1 - \beta} = \sum_{i=-M}^{0} \beta^{-i} \exp(-\frac{L(\pi_i)}{\gamma_i}) \left( 1 - L(\pi_i) \left( \frac{1}{\gamma_0} - \frac{1}{\gamma_{ii}} \right) \right)
\]

\[
= A - \frac{1}{\gamma_0} B + \exp(-\frac{L(\pi_0)}{\gamma_0}) = A - \frac{1}{\gamma_0} B + \exp(-\frac{L(\pi_0)}{\gamma_{-1}}) \left( 1 - L(\pi_0) \left( \frac{1}{\gamma_0} - \frac{1}{\gamma_{-1}} \right) \right)
\]

(4.3)

It is seen that the implicitly defined constants in (4.3) maintains the history of Boltzmann function values and temperatures. An approximation of \( \gamma_0 \) is obtained from (4.3) and we arrive at the following scheme to compute the current temperature for each arriving agent:

\[
\gamma_0 = \frac{B \cdot \exp(L(\pi_0)/\gamma_{-1}) + L(\pi_0)}{1 + \frac{L(\pi_0)}{\gamma_{-1}} + \exp(L(\pi_0)/\gamma_{-1}) \left( A - \rho \frac{1 - \beta^{M+1}}{1 - \beta} \right)}
\]

\[
A \leftarrow \beta A + \left( 1 + \frac{L(\pi_0)}{\gamma_0} \right) \exp(-L(\pi_0)/\gamma_0)
\]

\[
B \leftarrow \beta B + L(\pi_0) \exp(-L(\pi_0)/\gamma_0)
\]

\[
\gamma_{-1} \leftarrow \gamma_0
\]

\[
M \leftarrow M + 1
\]

where the initial values are \( A = B = M = 0 \) and \( \gamma_{-1} = -L(\pi_0)/\ln(\rho) \).

Similarly, after having updated the current temperature of its destination node, the agent backtrack along its path \( \pi_0 \) and update the probabilities \( p_{0,rs} \) of taking the various routes according to an autoregressive stochastic counterparts of (3.6).

\[
p^{*}_{0,rs} = \frac{T_{rs}}{\sum_{\gamma_{s}} T_{rs}}
\]

(4.5)

\[
T_{rs} = \sum_{i=-M}^{0} I(\{r,s\} \in \pi_i) \beta^{-i} \exp(-\frac{L(\pi_j)}{\gamma_0})
\]

(4.6)
Due to the constant temperature regime of (4.6) we have the same infeasible storage and computing requirements as when solving (4.2). Thus again we assume $\gamma$ not to change radically and apply a second order Taylor expansion to each term, i.e. (4.6) is approximated by
\[ 0 \sum_{i=0}^{N} I(\{r, s\} \in \pi_i)B^{-i}\exp \left( \frac{L(\pi_i)}{\gamma_0} \right) \left( 1 - \frac{L(\pi_i)}{\gamma_0} \right) \left( \frac{L(\pi_i)}{\gamma_0} \right)^2 \left( \frac{1}{\gamma_0} - \frac{1}{\gamma_0} \right)^2 \right) \].

The second order expansion is used to better approximate the hyperexponential numerator and denominator of $p^*_{0, rs}$ also avoiding non-physical (negative) values of $T_{rs}$ in case of a rapid decay of the temperature. However, this may result in a non-physical increase of the approximation of $T_{rs}$ as $1/\gamma_0$ increases. Hence, when the derivative of the approximation above becomes positive, it is replaced by its minimum which yields:

\[ T_{rs} = I(\{r, s\} \in \pi_0)\exp \left( -\frac{L(\pi_0)}{\gamma_0} \right) + A_{rs} + \begin{cases} 
-B_{rs} \frac{1}{\gamma_0} + C_{rs} \frac{1}{\gamma_0} - \frac{B_{rs}}{2C_{rs}}, & \frac{1}{\gamma_0} \leq \frac{B_{rs}}{2C_{rs}} \\
\frac{-B_{rs}^2}{4C_{rs}}, & \text{Otherwise}
\end{cases} \quad (4.7) \]

where, as for the temperature, we have an autoregressive updating scheme for the parameters yielding the second order approximation:

\[ A_{rs} \leftarrow \beta A_{rs} + I(\{r, s\} \in \pi_0)\exp \left( -\frac{L(\pi_0)}{\gamma_0} \right) \left( 1 + \frac{L(\pi_0)}{\gamma_0} \left( 1 + \frac{L(\pi_0)}{2\gamma_0} \right) \right) \]

\[ B_{rs} \leftarrow \beta B_{rs} + I(\{r, s\} \in \pi_0)\exp \left( -\frac{L(\pi_0)}{\gamma_0} \right) \left( L(\pi_0) + \frac{L(\pi_0)^2}{\gamma_0} \right) \quad (4.8) \]

\[ C_{rs} \leftarrow \beta C_{rs} + I(\{r, s\} \in \pi_0)\exp \left( -\frac{L(\pi_0)}{\gamma_0} \right) \frac{L(\pi_0)^2}{2} \]

The initial values of (4.8) are $A_{rs} = B_{rs} = C_{rs} = 0$.

The next agent arriving at node $r$ will according to the unconditional probability of (4.5) depart towards node $s$, where the “pheromone” $T_{rs}$ are determined according to (4.7) and updated according to (4.8) in its return. This is detailed in Section 4.3.

4.2 Initialization and Selection Strategies

An initialization phase is needed to establish a rough estimate of the temperature $\gamma$ under the initial routing probabilities. These probabilities are chosen to be uniformly distributed, $p_u$, which is similar to [6]. During this phase, the parameters of the autoregressive temperature computations in each node, i.e. (4.4), are obtained as well as initial values of the pheromone parameters of (4.8). The number of agents completing a tour
during the initialization is \( D - n \cdot m \) where \( n \) is the number of nodes in the network and \( m \) is the average number of outgoing links per node. The convergence of the algorithm is robust with respect to the initial routing probabilities and number of agents.

The actual next hop probability the agents uses, \( q_t \), must take into account the previously visited nodes. Both during the initialization phase and the rest of the search process our agents use the following selection strategy of the next node:

\[
X_{t,r}(s) = \begin{cases} 
1, & \text{if node } s \text{ has not already been visited} \\
1, & \text{if all nodes have been visited and } s = \text{homenode} \\
0, & \text{otherwise}
\end{cases}
\]

In networks that are not fully connected, an agent may experience that \( \sum_{s} X_{t,r}(s) = 0 \), i.e. it is stuck. In this case the agent is terminated.

After the initialization phase there will be a non-zero probability that \( X_{t,r}(s) \) may cause the vector \( q_{t,r} \) to be zero, i.e. all feasible routes are found to be inferior. When such a no-next-hop event occurs \( p_t \) is replaced by \( p_a \). By introducing a small noise component \( \varepsilon \), \( p_a \) is generated both during the \( \gamma \)-initialisation phase and when a no-next-hop events occur as shown in (4.9).

\[
q_{t,rs} = \frac{I(t > D)p_{t,r,s}(1 - \varepsilon) + \varepsilon X_{t,r}(s)}{\sum_{k} I(t > D)p_{t,r,k}(1 - \varepsilon) + \varepsilon X_{t,r}(k)} \quad \text{when } \sum_{s} X_{t,r}(s) > 0 \quad (4.9)
\]

where \( I(\ldots) \) is the indicator function and \( \varepsilon \) is chosen very small, e.g. \( 10^{-60} \).

The parameter \( \rho \) in (3.3), (4.2), etc. governs the emphasis put on the shorter routes. In [6] it is proposed to introduce an adaptive \( \rho \), i.e. \( \rho \) is decreased during the search process, which resulted in a slightly faster convergence. Our experiments show that our mobile agent algorithm converge significantly faster if \( \rho \) is decreased by 5\% when no improvement in minimum path cost has been observed after \( D \) tours. Decreasing \( \rho \) by a higher factor did not improve the convergence significantly but a lower factor reduced the speed-up notably. Hence, each agent home node performs the operation

\[
\rho \leftarrow 0.95 \cdot \rho, \ l \leftarrow i \ \text{when} \ (i - l = \lceil D/k \rceil) \land (\min_{j \leq l} (L(\pi_j)) = \min_{j \leq i} (L(\pi_j))) \quad (4.10)
\]

where \( k \) is the number of agent home nodes in the network.

### 4.3 Agent Behaviour Algorithm

Fig. 4.4 shows pseudo-code describing the behaviour of a mobile agent implementing our algorithm. Each node in the network is assumed to store the autoregressive parameters required by (4.8), its own address (current_node.address) and a minimum cost observed (current_node.L_min). The address is set when the network topology is created. The minimum cost is updated by agents visiting the node as shown in Fig. 4.4. according to (4.10), and is later used to trigger adjustments of the search focus parameter \( \rho \) described in Section 4.2.
Each node acting as a home node must in addition to the parameters required by (4.8) store autoregressive parameters required by (4.4).
No synchronisation between agents takes place during a search scenario. Only indirect communication is performed by accessing path quality marks \((p_i)\) and the propagated minimum cost value.

Each agent starts every search for a path from its home node. Agents with the same home node cooperate in adjusting the temperature stored in the node. Thus a range of search scenarios are possible where one extreme is having all agents share the same home node and another extreme is letting each agent have its private home node. In Section 6 we examine simulation results from both extremes.

5 Implementation in the Network Simulator

Due to the stochastic nature of our mobile agents it is difficult to predict the exact behaviour demonstrated. The behaviour of a single agent is to some extent trackable but when a number of agents are executing concurrently and asynchronously the overall system behaviour becomes too complex for formal analysis which leaves us with the option of collecting results using Monte Carlo simulations.

Instead of designing and implementing a complete simulator with configurable environmental parameters we chose to enhance an already well tested open source simulator package, the Network Simulator (NS) [14]. NS is capable of running realistic simulation scenarios of traffic patterns in IP-based networks. Dynamic topologies both wireline and wireless, miscellaneous protocols and a collections of traffic generators are supported. The package is implemented as a mix of OTcl and C++ classes.

We have made the NS-package capable of handling mobile code simulations by adding functionality for Active Networking (AN) [8, 15]. The extension is based on work done in the PANAMA project (TASC and the University of Massachusetts).

Fig. 5.1 illustrates how some of the environmental objects and a mobile agent interact during a simulation. A Tcl simulation control object creates node and mobile agent kernel objects (C++). The new kernel objects are controlled through Tcl-mirror objects but

![Fig. 5.1 Schematic representation of interactions between objects in the NS simulator.](image-url)
to avoid unnecessary overhead during simulations only infrequent operations (e.g. initialisation) are executed through this interface. The numbered message sequence illustrates how a mobile agent is transferred between two active network enabled nodes. For performance reasons only references (and size info) are passed between the nodes.

6 Case studies

We selected four different topologies from TSPLIB [16] to demonstrated the performance of our algorithm. Three of the topologies where selected specifically such that a comparisons between our algorithm, Rubinstein’s algorithms and the Ant Colony System could be performed. Table 6.1 shows the results.

Table 6.1 Lists results from nine different simulation scenarios. By default all agents in our algorithm has different home nodes. Scenarios marked with * in the left column are exceptions where all agents have the same home node.

<table>
<thead>
<tr>
<th>Topology</th>
<th>No of nodes</th>
<th>No of agents</th>
<th>No of tours</th>
<th>Best tour</th>
<th>Converged average</th>
<th>Rubinstein total no of samples</th>
<th>Rubinstein best tour</th>
<th>ACS best tour</th>
<th>Best known (TSP-LIB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>fri26</td>
<td>26</td>
<td>1</td>
<td>133 267</td>
<td>960</td>
<td>1010 (± 18)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>937</td>
</tr>
<tr>
<td>fri26</td>
<td>26</td>
<td>26</td>
<td>358 564</td>
<td>940 (994)</td>
<td>970 (± 20)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>fri26*</td>
<td>26</td>
<td>26</td>
<td>149 453</td>
<td>943 (1077)</td>
<td>1022 (± 34)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ry48p</td>
<td>48</td>
<td>1</td>
<td>308 401</td>
<td>15 169</td>
<td>15 725 (± 340)</td>
<td>172 800</td>
<td>15509</td>
<td>14422</td>
<td>14422</td>
</tr>
<tr>
<td>ry48p</td>
<td>48</td>
<td>48</td>
<td>947 063</td>
<td>14 618</td>
<td>15 139 (± 212)</td>
<td>238 765</td>
<td>7111</td>
<td>-</td>
<td>6905</td>
</tr>
<tr>
<td>ft53p</td>
<td>53</td>
<td>1</td>
<td>311 555</td>
<td>7 351</td>
<td>7 702 (± 249)</td>
<td>1120000</td>
<td>39712</td>
<td>36230</td>
<td>36230</td>
</tr>
<tr>
<td>ft53p</td>
<td>53</td>
<td>53</td>
<td>1 270 621</td>
<td>7 122</td>
<td>7 487 (± 209)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>kro124p</td>
<td>100</td>
<td>1</td>
<td>579 618</td>
<td>40 807</td>
<td>43 128 (± 1167)</td>
<td>1120000</td>
<td>39712</td>
<td>36230</td>
<td>36230</td>
</tr>
<tr>
<td>kro124p</td>
<td>100</td>
<td>100</td>
<td>3 359 288</td>
<td>38 352</td>
<td>40 095 (± 2054)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Column 2-6 (counting from the left) show parameter settings and performance results from our distributed algorithm. Column 1 and 2 gives the name of the topology used in the scenario and the number of nodes of the topology. Column 3 shows the number of agents and autonomous home nodes applied in parallel during simulation. Column 4 shows the total number of tours traversed before all agents converged towards a tour with the cost given in column 6. Column 5 shows the best tour found. Column 4 and 6 are averaged over 12 simulations with standard deviation shown in brackets while column 5 is the best of the best values found among 12 simulations with the worst of the best in brackets.

Column 7-9 show results obtained by two centralized algorithms, Rubinstein’s original algorithm and the Ant Colony System version Opt-3. The last column shows the best known results listed in TSPLIB.

Empirically we found that the following parameter settings gave good results:
\[ \beta = 0.998, \rho = 0.01 \text{ and } \rho\text{-reduction factor} = 0.95 \]. Thus they have been applied in all the simulation scenarios.

In general our algorithm finds good solution to the TSP scenarios tested, close to (and in a few occasion better than) the results reported by Rubinstein. But speed of convergence is not equally good. Our algorithm requires up to 5 times more tours to be traversed before convergence compared to the total number of samples in Rubinstein’s algorithm. (Rubinstein stops his algorithm when the best tour found has not improved in a certain number of iteration. We report the number of tours required for all agents to converge towards one path. Results can still be compared since best tours for our algorithm are in general found only a short while before full convergence.) Small standard error values indicate stable convergence over several simulation runs.

When comparing results from scenarios run on the same topology but with a different number of agents we observe that our algorithm requires a higher number of tours for scenarios where multiple agents are searching in parallel than for single agent scenarios. Still the number of tours per agent is significantly less for the multi-agent cases, close to 10 times less for fri26, 20 times less for ry48p, 15 times less for ft53p, and 15 times less for kro124p. In a true concurrent environment this would result in the respective real time performance gains.

In the scenario named fri26* 26 agents search in parallel and share the same home node, i.e. they share the same set of autoregressive parameters required by (4.4). They use approx. the same total number of tours to converge as in the single agent version of fri26. Thus real time performance is improved by a factor equal to the number of agents. However the converged average is higher than for the other fri26 scenarios, i.e. premature convergence is more common. Having only a single home node also introduces a single point of failure which contradicts with our objective of a dependable distributed system.

The ACS-opt3 algorithm is implemented as a complex mix of iterations using heuristics for local optimization and iterations using global optimization by pheromone tails. Thus it is difficult to compare performance results by other means than best tour found and CPU time required. Our simulator is not implemented with the objective of solving
TSPs as fast as possible. Thus CPU time is no good performance measure which leaves best tour as the only comparable result. The ACS-opt3 algorithm finds better best tours than both our algorithm and Rubinstein’s.

7 Concluding remarks

In this paper we have introduced an algorithm for solving routing problems in communications networks. The algorithm is fully distributed and well suited for implementation by use of simple autonomous mobile agents encapsulated in for instance active network packets. Agents act asynchronously and independently and communicate with each other only indirectly using path quality markings, (pheromone trails) and one shared search control parameters.

In contrast to other “ant-inspired” distributed stochastic routing algorithms our algorithm has a mathematical foundation inherited from Reuven Rubinstein’s cross-entropy method for combinatorial optimization [6]. Rubinstein propose an efficient search algorithm using Kullback-Leibler cross-entropy, important sampling, Markov chains and the Boltzmann distribution. However his algorithm is centralized and batch oriented. By introducing autoregressive stochastic counterparts to Rubinstein’s method of shifting routing probabilities we have removed the need of centralized control. In addition, due the necessary approximations made, we have reduce the computational load of handling an agent in a node to a few simple arithmetic operations.

Performance wise the new algorithm shows good results when tested on a hard (NP-complete) routing problem, the Travelling Salesman Problem. Compared to Rubinstein’s algorithm up to 5 times more paths need to be tested before convergence towards a near optimal path takes place. Increasing the number of agents searching in parallel decrease significantly the number of tours per agent required to find a high quality path.

No excessive parameter tuning has so far been performed. Further investigation is required specially on the effect of adjusting the weight put on historical information (β) during the search process. Pros and cons of making more (or less) global knowledge available (i.e. let more parameters values propagated throughout the network) should also be looked into.

Currently new versions of our algorithm is under development where heuristic techniques found in algorithms like the Ant Colony System [9] are incorporate to improve performance. Additionally, by altering the search strategies, we expect our algorithm to find optimal tours when network topologies are far from fully meshed (as it is for TSPs) and do not allow all nodes to be visited only once.

Other ongoing work includes having several species of agents compete in finding quality paths in a network. Early results indicate that a set of disjunct high quality paths can be found efficiently. We intend to investigate the applicability of such a system to the routing problems encountered by Grover in his work on restorable network and protection cycles [17].
References


Context Gathering Mobile Agents to Assist in Resource Reservation for Inter-technology Hand-off

N. S. Satish Jamadagni
Research Scientist, R&D
SASKEN communication technologies Ltd
No 3008, 12th B Main, 8th Cross, HAL 2nd stage, Indiranagar
Bangalore 560 008, India
satishj@sasken.com

Abstract. Mobility between dissimilar networks is one of the trends in network design with the availability of multi-mode terminals moving towards fourth generation telecommunication systems. Overlaid networks provide support to such multi-mode terminals in an efficient and scalable way. Handoff support for multi-mode terminals under overlay networks involves dynamic resource reservations to guarantee smooth handoff. Wireless overlay networks present new problems in achieving efficient resource reservation for handoff support. Signaling protocols to support handoff between such heterogeneous standards might take a long time to evolve and can be insufficient. Such signaling might not be elaborate to facilitate efficient resource reservations to support inter-technology handoff. In this paper, we suggest the use of mobile agents that travel through the Internet backbone across system boundaries to establish a communication channel between heterogeneous wireless systems and specifically help in usage context gathering at the boundary of overlaid wireless networks for achieving efficient resource reservations.

1 Introduction

Third generation networks such as the IMT 2000 and the UMTS promise heterogeneous multimedia-based services to users who may roam across various tiers, regions and networks. Such future mobile terminals may operate in multiple modes with separate transmitter/receiver pairs, such as the satellite/terrestrial multi-mode terminals, or the MTs may be reconfigured to operate in each new system. It is also assumed that such multi-mode terminals can measure and compare signals from different air interfaces and power levels. Roaming between systems requires that the radio network system support handoff between these different types of networks. Handoff management allows a call in progress to continue as the mobile terminal (MT) changes channels or moves between service areas. Handoff criteria for inter technology handoffs can be QoS availability, cost of network access, service
availability in a geographic location, traffic conditions etc. A cost factor example for inter-technology handoff is a handoff from a satellite system to a terrestrial system, access through satellite systems being costly. It is essential to minimize handoff delay during handoff as this could lead to call drops. A factor that affects fast handoff is the ready availability of channels or resources in a system to accommodate mobile terminals (MTs) coming into a system. In this paper we address the issue of achieving optimal resource reservation to accommodate inter-system handoffs. A network system should be aware of the mobile terminal usage context in surrounding systems to facilitate proper resource reservation for handoff.

There are several special cellular architectures that try to improve spectral efficiency and accommodate handoff issues with out a large increase in infrastructure costs. Some of these structures include underlay/overlay systems and multi-channel bandwidth systems [3]. In multi-channel bandwidth systems, a cell has two or three regions with different bandwidth channels. The specific topology of cells and the wide variety of network technologies that comprise wireless overlay network present new problems that have not been encountered in previous handoff systems. Some of the inter-technology handoff scenarios are summarized below (with system A being a wide area low bandwidth cellular system Ex: GPRS and system B being a local area high bandwidth system Ex: WLAN or Bluetooth).

Moving from system A to system B (cellular to WLAN)
Moving from system B to system A (WLAN to cellular)
Moving through – a combination of the above two.

Monitoring use context of mobile devices in the vicinity of a wireless technology overlap can help make prediction on handoff. This also helps support handoff through resource reservation decisions. For inter-technology handoff the Mobile Terminal (MT) must be aware of the existence of the boundaries between the two technologies, basically this would involve that the MT be aware of its own location. Location information can be captured in distributed databases called secure coordinate servers as in [1]. Location information alone is insufficient in achieving efficient reservation schemes. To facilitate seamless handoff with out service disruption different networks systems should have an estimate of the number of handoffs and the Quality of Service (QoS) requirements of mobile devices coming in from different network technologies.

One of the important considerations in achieving seamless inter-technology handoff would be to arrive at a good radio resource management scheme to accommodate handoffs from other systems. Radio resource management tasks performed by wireless networks include admission control, channel reservation and assignment, power control and handoff. An integrated radio resource management scheme can make necessary tradeoffs between the individual goals of these tasks to obtain better performance. The awareness of a mobile usage context basically helps in estimating the probability of handoff and thus helps in making resource reservations. Inter-technology handoffs are basically treated as new calls in the new system as the mobile terminal seeks a new link in the new system. Inter-technology handoff should also allow for notifying the new system that a new call being established by a multi-mode terminal has to be treated preferentially and not as a new call.

In this paper, we suggest the use of push and pull of lightweight mobile agents at the boundary of the overlaid wireless network technologies to gather context information that would assist in resource reservations for inter-technology handoff.
Mobile agents are programs that can migrate from host to host in a network, at times and to places of their own choosing. The state of the running program is saved, transported to the new host, and restored, allowing the program to continue where it left off. Mobile agents are an effective choice for many applications including improvements in latency and bandwidth of client-server applications and reducing vulnerability to network disconnection. Agents even if they are performing simple tasks, can achieve a significant performance improvement by moving themselves to more attractive network locations.

2 Inter Technology Handoff

Next generation wireless communication is based on a global system of fixed and wireless mobile services that are transportable across different network backbones, network service providers, and network geographical boundaries. There are two types of roaming for the mobile user: intra-system roaming and inter-system roaming. Intra-system roaming refers to MTs that move between different tiers of the same system, i.e., between the pico, micro, and macro cells of the same network. Inter-system roaming refers to MTs that move between different backbones, protocols, or service providers. Handoff can be performed using three types of control methods: Network-Controlled Handoff (NCHO), Mobile-Assisted Handoff (MAHO), or Mobile-Controlled Handoff (MCHO). Under NCHO or MAHO the network generates the new connection, finding new resources for the handoff and performing any additional routing operations. For MCHO handoff, the mobile terminal finds the new resources and the network approves.

Several architectures are emerging in support of inter-system handoff and one of them is through boundary region cells being considered for the next generation wireless systems. It is illustrated in figures-1 below. The boundary region consists of inter-system boundary cells that lie in the overlap area between two networks. Each boundary cell is generally controlled by a boundary cell base station, which is connected to a switch or router in its own network, as shown in the Figure. While inside one of the boundary cells, the MT is able to transmit and receive broadcast signals from either network, depending on the MT's current configuration. Signaling and control messages passed between the boundary cell base stations and their network switches could reroute the MT's connections before the MT handed off into the new system. Such architecture depends on an explicit signaling protocol in the boundary cells to support inter-system handoff. It also insists that multiple physical network interfaces should be open in the boundary region to facilitate handoff. This might be inefficient in terms of battery power consumption.

Although Figure-1 shows the inter-system boundary region as a physical area between two networks, the inter-system boundary can be designated as a virtual region between any numbers of networks. For example, an urban area may be expected to have overload conditions during rush hours, users would be able to switch their service between pico-, micro and macro-cell tiers in the terrestrial network, or switch from terrestrial to the satellite network.
An inter-system handoff signaling protocol thus defined focuses on the handoff signaling procedures and not necessarily support dynamic resource reservation decisions. The signaling protocol supporting inter-technology handoff should allow for enough details so that the system accepting the mobile terminal can make proper resource reservations in anticipation of handoffs. With such elaborate signaling lacking between different technologies there is a need to gather the usage context of mobiles in the boundary regions for resource reservation purposes.

3 Mobile Agents for Mobile Terminal Context Gathering

In this paper we use mobile agents to gather the mobile terminal QoS context information based on which resource reservation decisions in support of handoff with QoS constraints are made. This leads to better resource reservation schemes in the
overlaid systems. The performance benefits will be in terms of optimal resource utilization. The use of mobile agents include reduction in network bandwidth consumption, reduced latency, reduced computation and increased fault tolerance. The use of mobile agents is also motivated by the following:

There might not exist an inter technology signaling standard that would facilitate handoff and the use of mobile agents to detect the existence of such networks and facilitate handoff becomes inevitable.

Even if a signaling protocol exists – the continuous monitoring of a set of devices (profile and context exchange or update) and their QoS requirements could be unnecessarily heavy on the base-station. The sufficiency of such protocols is also a major concern.

The Figure below depicts the use of the internet backbone for pushing context gathering agents into the transition region of an overlay network. A transition region is defined as a region where the possibility of handoff is quite high. A more detailed definition is not within the scope of this paper.

![Diagram of network backbone](image)

**Figure 3 – Agents pushed into the MTs in the transition region to gather context information**

### 3.1 The Mobile Terminal Architecture

The use of mobile agents to gather context information for effective resource reservation in support of inter-system handoff requires proper system support. Our agent infrastructure assumes the use of Hypertext Transfer Protocol (HTTP) for agent transfer and communication [15]. The choice of a HTTP based agent support system was also driven by the fact that the WAP forum is advocating HTTP stack on a mobile terminal to support PUSH. We also expect future advances in, e.g., HTTP security and electronic payment resulting from the World Wide Web research community to save considerable effort, which would otherwise be necessary to implement such in some separate framework for mobile agents.

On the system side (basestation) context aggregation, context agent selection and dispatch mechanism are controlled. The decisions on when to push agents with what query mechanism is supported by an agent is dictated by the QoS parameters supported by a protocol. Open interfaces are assumed at the mobile terminal side to
support QoS and other parameter query. It is also assumed that the mobile terminals support HTTP based PUSH [8] protocol so that context agents can be pushed onto the mobile terminals.

3.2 The Context Agent

There has been a proliferation of communication devices with varying capabilities in recent days. Devices can be supporting a variety of browsers, CPU, memory size, display screen size, scripting language support and application support. Resource reservations and QoS considerations will have to be aware of the individual user terminal capabilities as well usage context for optimized handoff reservations. This highlights the need for the context agent to have a schema to capture user context. The context agent used in our experiments use a minimal set of context variables for monitoring that is sufficient for resource reservation decisions. The list of variables is provided below. Also the context agent embeds algorithms to aggregated information about the usage context of a mobile terminal. The context agent with different algorithms are identified by an identifier. The contextual variants are listed below.

<table>
<thead>
<tr>
<th>Table – 1: Context variables</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Context variables monitored by the context agent</strong></td>
</tr>
<tr>
<td>&lt;Location, approximate speed, direction of travel etc&gt;</td>
</tr>
<tr>
<td><strong>Applications</strong></td>
</tr>
<tr>
<td>&lt;App1 – network connectivity demands–recent usage count&gt;</td>
</tr>
<tr>
<td>&lt;App2 – network connectivity demands–recent usage count&gt;</td>
</tr>
<tr>
<td><strong>QoS parameters – QoS service class</strong></td>
</tr>
<tr>
<td>&lt; Latency tolerance, packet loss tolerance etc&gt;</td>
</tr>
<tr>
<td>&lt;Application Loss sentivity and Latency sensitivity&gt;</td>
</tr>
<tr>
<td><strong>Personalized services</strong></td>
</tr>
<tr>
<td>&lt;active services - stock price tracking, news etc&gt;</td>
</tr>
<tr>
<td>&lt;Virtual home environment settings&gt;</td>
</tr>
</tbody>
</table>

Fuzzy values of the above mentioned variables are gathered by the context agent implemented in our experiments. Agent gathered fuzzy knowledge is represented as

$$K_a(f) = \mu_f(a), a \in A, f \in F$$

where

$$\mu_f(f) : F \rightarrow [0,1], f \in F$$

Further we use Fuzzy Cognitive Maps (FCM) [14] in the context agent to map the context variables listed above to inter-system handoff resource reservation. FCMs are signed directed graphs, where nodes stand for context variables and the edges stand for the partial causal flow between the nodes and handoff belief value. Generally an FCM stores a set of rules of the form “If Pos(A) then possible sequence A”.

Context agents are system launched i.e. mobile agents are launched by the system to gather the context information of mobile users. System launched agents can be simple agents with out complex context aggregation algorithms or can also be embedded with complex aggregation algorithms. Context aggregation algorithms are
useful to arrive at the resource reservation requirements in the new system. Aggregation algorithms cover the mobile terminals that are in the transition region. The system can Push or Pull mobile agents into and from a given geographic location [4]. Multicast methods can also be used to Push the agents to a specified number of mobile terminals and Pulled back after some time. The number of agents launched is again left to the system which can follow a specified algorithm. The push region is adaptively arrived at with feedback on the actual handoff of those terminals and the location of the MTs with respect to the boundary region.

Results and Conclusions

In our experiments we have considered a vertical inter-system handoff between a WLAN (IEEE 802.11) and GPRS in a simulated environment. We have simulated random number of mobile terminals in a transition region between the enterprise wireless LAN and the wide area GPRS network. Our initial experiments were limited to demonstrating the effectiveness of using mobile agents in gathering context information for efficient resource reservations in the presence of an insufficient inter-system handoff signalling. The use of mobile agents also poses realtime constraints that are considered. We study the effectiveness of using mobile agents for context information gathering for resource reservation and the regular inter-system handoff between WLAN and GPRS [13].

The handoff between WLAN to GPRS is considered as this poses severe degradation of service experienced by the mobile terminal moving from a higher data rate WLAN to the lower data rate GPRS system. To support such a handoff the GPRS network is under strain to ensure a dynamic resource reservation scheme to support seamless handoff. The tendency of the mobile terminal will be to hang on to the WLAN system as long as possible and the GPRS resource reservation schemes should also have an estimate on the number of mobile handing off into the system. Channel reservations by GPRS system is critical as the mobile terminal would prefer a very fast handoff to maintain higher layer connectivity. We consider a random mobility model where not all the mobile terminals in the transition region will require handoff. We use mobile agents with predictive fuzzy cognitive maps to assert the handoff belief.

![Figure-4](plot.png)

**Figure-4** Plot showing the sufficiency of channel reservations decided due to context agents

![Samples at different time intervals](chart.png)

- **Mobiles entering Tr**
- **Channels reserved**
- **No of actual handoff**
We have considered only the Location, approximate speed and application QoS service class to achieve reservations. We have also not considered the delay involved in mobile agent travel across the internet backbone. Work in these directions would further prove the importance of mobile agents in inter-system handoff.

References

1. M Ylianttila, J Makela and K Pahlavan, Geolocation information and intertechnology handoff, 0-7803-6283-7/00 IEEE.
4. H Farooq Ahmed, Kinji Mori, Push and Pull of information in autonomous information service systems, 0-7803-6575-5/00 IEEE
5. Ravi Jain and Farooq Anjum, Mobile agents for personalized information retrieval: When are they a good idea, 0-7803-6596-8 IEEE.
12. Resource Description Framework (RDF), Model and syntax specification, W3C working draft.
15. Anselm Lingnau, Oswald Drobnik, Peter Dömel, An HTTP based infrastructure for mobile agents - http://www.w3.org/Conferences/WWW4/Papers/150/
Towards a Secure Agent Platform Based on FIPA

Min Zhang¹, Ahmed Karmouch¹, Roger Impey²

¹Multimedia & Mobile Agent Research Laboratory
School of Information Technology & Engineering (SITE)
161, Louis Pasteur St. Ottawa (ON), K1N 6Y5, CANADA
Email: mzhang@sol genie. uottawa.ca; karmouch@site.uottawa.ca

²Institute for Information Technology, National Research Council of Canada
Ottawa (ON), K1A 0R6, CANADA
Email: Roger. Impey@iit. nrc.ca

Abstract. FIPA specification enables the interoperability among a diversity of agent platforms in a highly heterogeneous computing environment. Agents of different systems or providers, as far as they are all FIPA-compliant, can communicate and interact directly by Agent Communication Language (ACL). However, potential security threats in agent platforms are not fully addressed in both the FIPA specification and most of its implementations such as FIPA-OS. In order to add security features to FIPA, we propose a two-layer architecture that includes a security layer as the security extension to FIPA-OS. This architecture provides two types of security-related services to agents: a secure communication service which prevents any eavesdropping or interference from the outside network, and a secure execution environment service which protects server resources and agent services from any unauthorized access of agents. In this paper we present the design and implementation of this architecture as well as the trust model.

1 Introduction

Agent technology prompts current computing environments to be highly distributed and heterogeneous. Each agent is an independent computation unit acting on behalf of its owner. Unlike stationary agents, which may communicate with other agents remotely, a mobile agent can migrate from its home site to another site, “talking” with other agents locally by peer-to-peer communication. Because of advantages such as the reduction of network traffic and enabling systems to be more autonomous, mobile agent technology is receiving extraordinary attention and has been proposed in a variety of application fields, such as electronic commerce [1], network management [2], distributed information retrieval [3].

There is a diversity of agent platforms or agent servers that have their own platform-specific service ontology, encoding mechanisms and communication protocols, e.g. Agent Tel [4], Concodia [5], and Voyager [6]. Though well designed, most of these platforms do not support interoperability, and this platform exclusion
restricts the propagation of agent technology. In 1997, the development of two standards, Mobile Agent System Interoperability Facility (MASIF) [7] and Foundation for Intelligent Physical Agent (FIPA) [8], have changed this situation greatly. MASIF is a set of interfaces and basic technologies that can be integrated by developers to develop complex systems with a high degree of interoperability. FIPA is more like an abstract architecture that can be shared by different platform implementations, and agents of different systems or providers, as far as they are all FIPA-compliant, can communicate and interact with each other directly by Agent Communication Language (ACL).

There are now several agent platforms that claim to support FIPA standard, including FIPA-OS [9], JADE [10], Grasshopper [11], and ZEUS [12]. FIPA-OS (FIPA Open Source) is used in our implementation as the underlying platform.

In this paper, we propose a security architecture based on FIPA-OS together with the security services it provides. The rest of the paper is organized as follows: Section 2 gives an overview of the security threats present in the mobile agent system. In section 3, we describe the two-layer security architecture and our trust model, which consists of assumptions of trust relationships among entities. In sections 4 and 5, the secure communication service and the secure execution environment service are described in detail. Finally we draw our conclusions and future work in section 6.

2 Security Threats in the Mobile Agent System

Security threat is one of the challenges preventing mobile agent systems from being more widely deployed. The reason for this lies largely on one intrinsic characteristic of mobile agents: execution on remote unknown platforms rather than their safe home sites, especially in a heterogeneous and open computing environment like the Internet. Unless countermeasures are taken, both agents and agent platforms are vulnerable and need certain types of protection. Figure 1 shows the attack model of a mobile agent system. The potential threats in this model can be classified into four categories: (1) Agent against agent platform, (2) Agent platform against agent, (3) Agent against agent, and (4) Outside network against agent.

![Fig. 1. Attack Model of a Mobile Agent System](image-url)
Security attacks can be carried out in both passive and active ways, typically the breach of privacy, damage or destruction, masquerading, denial of service, harassment, unauthorized access, etc. In the threats of agent against agent platform, an agent is trying to destroy the server host or gain unauthorized access to the agent platform. Passive attacks include communications monitoring and sensitive information pilfering. Active attacks include the damage of the host’s resources via deletion or modification. In the category of agent platform against agent, the server may try to tamper with the agent or extract sensitive information (such as credit card numbers in E-Commerce applications) out of the agent without the agent’s acknowledgment or detection. It is more difficult to prevent these attacks because the host has full access to the mobile code and its data, therefore the agent cannot be effectively protected against interception and tampering. As Chess argued in [17], it is almost impossible to prevent these attacks without the presence of the tamper-proof hardware. Besides the threats from agent platform, mobile agents are vulnerable to attacks from neighboring agents as well when they arrive at the destination host. Agents may exploit security weaknesses of other agents or launch attacks against other agents. The last category of threats is that of outside network against agent. A mobile agent is under risk from the outside network when it is migrating or communicating with its home site. The typical attacks include passive ones such as eavesdropping and traffic analysis, and active ones such as message modification and forging.

3 Security Architecture

3.1 Two-Layer Structure

![Two-Layer Structure Diagram]

**Fig. 2.** Two–layer structure
In this paper we propose the architecture of a two-layer agent platform in order to add security features to the FIPA. A FIPA-OS agent platform is employed here as the agent management and communication infrastructure, while a separate security layer is designed as a security extension to the FIPA-OS and provides security-related services to agents. The secure communication service and the secure execution environment service are provided respectively to address the security requirements known as Communication Security (type A in fig 1) and Computation Security (type B). The FIPA-OS has implemented mandatory elements contained within the FIPA-97 specification and also supports agent interoperability defined in that specification. Figure 2 roughly illustrates this architecture. The security layer in an agent platform ensures security in the format of security services.

The relationship between these two layers is twofold: on the one hand, the security layer relies largely on the underlying agent platform in communication and agent management. For instance, major components of the security layer such as the Secure Agent Channel Communication (SACC) and the Credential Granting Center (CGC) are agents registered in the AMS of the local FIPA-OS agent platform. The SACC implements its secure communication service partly by requesting for service provided by the ACC in the FIPA-OS. On the other hand, the separation of the security layer from FIPA-OS frees security mechanisms and policies from specific agent platforms. Only services defined in the FIPA specification are used by components in the security layer. That is, the security layer only involves functions of the underlying agent platform rather than the details of its implementation techniques.

Thus the principle benefit that derives from the separation of the security layer and the basic agent platform layer is the independence of security mechanisms. Different platforms with specific security mechanisms are able to interact under the same security architecture. Hence, the design of this architecture is more focused on the functions of a security layer, than on specific techniques in the implementation of the security layer.

The motive behind this two-layer architecture is manifold. Firstly, as mentioned before, the FIPA aims at the integration and high degree of interoperability of different platforms. The security specifications in FIPA-97 and FIPA-98 have been declared obsolete in recent FIPA-2000, and there is currently no agent security architecture defined in FIPA. This architecture is designed to provide security services and be compliant with the goal of the FIPA as well. Secondly, we all know that the security preparation or cryptographic calculation consumes a lot of time and computation power. For the reason of efficiency, some agents may wish to neglect security services for unimportant messages. This two-layer architecture helps provide both security services and basic agent platform services simultaneously, balancing the importance of the message and its efficiency requirements.

### 3.2 Trust Model

One category of security threats in figure 1 that is not considered in our architecture is the risk of the malicious agent platform. "Because of a lack of effective methods, the functional components in both the security layer and the basic layer in the agent platform, e.g. ACC, AMS, DF, SACC and CGC, as well as the internal message communication of a platform, are all regarded as safe. However, slightly different
from the assumption in FIPA specification, as discussed in [23], we assume that only those platforms known to a Certificate Authority (CA) are trustworthy. In other words, we consider a platform that is unknown or is without adequate proof from the CA (digital certificate) to be susceptible to other platforms, which may refuse to send critical messages and may prevent the migration of agents to this platform. Hence the CA is the root of all trust relationships in our system.

Our system relies on a set of baseline assumptions regarding the trust relationship of all entities including agent platforms, agents and users. These assumptions constitute the trust model in our system as illustrated in figure 3, which can be described as following:

- The CA is reliable to all entities within the system’s scope: the CA is the root of trust, thus all certificates issued by the CA are trustworthy.
- Users and agent platforms with genuine certificate are trustworthy: the CA will issue certificates only to those principles it knows very well and whose intention and deeds can be promised to be harmless. Thus, principles with certificate are trustworthy.
- The agent that are signed by trustworthy principles are trustworthy: the principle who has signed an agent, whether mobile or stationary, has a full knowledge of its workflow and behavior, and thus all the possible consequences of its execution. In other words, a trusty principle would never sign an agent which might do harm to other agents or the agent platform. The principle will be responsible for any agents that they have signed.
- The agent trusts its home agent platform: these platforms are not malicious. When an agent registers to its HAP, it assumes that its HAP is trustworthy, therefore it entrusts its profiles to it. This platform includes components in the FIPA-OS platform layer (e.g. ACC, AMS, DF), components in the security layer (SACC, CGC, Authenticator) and the internal communication channel.
Secure Communication Service

In the secure communication service, the security layer, mainly the Secure Agent Communication Channel (SACC), thwarts network communication based attacks, such as masquerading, eavesdropping, and tampering, by using authentication, encryption, integrity protection and replay detection. Here, cryptography is the key technology employed in our information security. Using the cryptographic modules, the SACC agent provides encapsulation and encryption service to incoming and outgoing messages, functioning as the secure communication gateway of its platform. Note here that the internal platform communication is regarded as safe in our trust model.

Two stages are involved in the preparation of this service in the security layer, a mutual authentication of platforms followed by a bilateral negotiation between platforms. After these two stages, point-to-point secure communication links are set up from one platform to each of its “adjacent” platforms (logically), or more precisely, from one SACC to the other. Based on the results of these two stages, a certain security level is accepted by both sides, which may be “low”, “medium”, “high” or “none”. It is only when these two stages are finished successfully, that the outgoing messages can be encrypted and then encapsulated in the security layer before being sent out, and recovered on the receiver platform.

In some situations, rather than a direct communication link between two platforms, an indirect link is used with the help of some intermediate platforms, e.g. when a certain security level requirement cannot be satisfied directly. However, an indirect message requires that the sender agent have full knowledge of all intermediate sites and indicate these sites explicitly in the message. Therefore more than two intermediate platforms are not feasible in practice. Note that when an agent uses secure services, from the agent’s perspective, the communication between two different platforms is via the Secure Agent Communication Channel (SACC) agent. However from the platform’s perspective all communications are still via ACC.

4.1 Mutual Platform Authentication

The primary goal of platform authentication is to verify each platform’s claim of identity, providing authenticity of platforms. Thus mutual platform authentication of agent platforms must be performed before secure communication service can be provided to agents. If the authentication with a platform fails, then any message with security requirements cannot be forwarded to that platform directly, and the security layer will inform the sender agent a “fail” message.

An authentication protocol can be based on a conventional secret key system or a public key system or both. Our authentication protocol is based on the latter, as well as a commonly trusted third party, the Certificate Authority (CA). In this protocol, each legitimate platform has its own private key and public key pair. As explained in fig 3, the CA signs certificates to verify both sides, which include platforms’ names and public keys. Therefore this authentication protocol is based on the exchange of certificates and verification of them.

In our authentication protocol, Since the certificates issued by the CA are signed by CA’s private key, those certificates are safe to be distributed. Therefore, the CA does
not play an active role in the protocol. Instead of requiring the partner’s certificate from the CA, each platform gets the certificate directly from its partner. Note that the process of sharing symmetric key is moved into the following negotiation protocol rather than in this authentication protocol. The reason for this is simple, because it is only when two parties agree on the same cryptographic options and security level that they can begin to exchange keys. The protocol is illustrated in the following figure 4:

![Authentication Protocol Diagram](image)

**Fig. 4. Authentication Protocol**

1. Platform A sends a “Hello” message to Platform B, initiating the process.
2. Platform B sends back its certificate (CertB) to platform A, including B’s name (B) and public key (PKB), signed by the CA’s private key (PRC), and an encrypted timestamp E(TB)PRB, encrypted by its private key (PRB).
3. Platform A verifies CA’s signature to confirm that the key it received is actually Platform B’s public key, then decrypt timestamp with the newly derived PKB. If ok, it sends its certificate (CertA) to platform B, including A’s name (A) and public key (PKA), also signed by the CA’s private key, with newly encrypted timestamp E(TA)PRA as well.
4. Platform B verifies CA’s signature to confirm that the key it received is actually Platform A’s public key. Then it decrypts timestamp TA with newly derived PKA to make sure that it is from platform A. Platform B then sends a reply message indicating the success and give the negotiation protocol that is to be followed.

If the partner platform is successfully authenticated, the platform will start the next step called the bilateral negotiation. Otherwise, this partner platform is marked “unknown” and the level of this secure link is marked “none”, showing that no secure communication service can be provided to this platform.

### 4.2 Bilateral Negotiation

The bilateral negotiation of platforms happens when the mutual platform authentication is successfully finished. Because they work mostly in a heterogeneous environment, the platforms in FIPA cannot be assumed to use cooperative negotiation
strategies designed centrally. Instead, each platform may use a strategy that provides the highest possible security configurations for itself without any concern for the other agent platforms' configurations. Therefore different platforms may have a diversity of cryptographic algorithms and support several security levels. In order to understand each other’s cipher message, two platforms must reach an agreement on the cryptographic options first. The bilateral negotiation protocol helps a platform to decide which option to be selected in respect to its partner’s cryptographic capabilities. Only the bilateral negotiation is considered because the inter-connection of all platforms is set up gradually by pairs of point-to-point secure connection. Two synchronous automatic negotiation protocols are provided: a preemptive negotiation protocol and a reactive negotiation protocol.

1. Preemptive negotiation: the preemptive negotiation is simple but efficient. In this protocol, the receiver can decide to either agree or not agree to the sender’s cryptographic capabilities and suggestions. Obviously, the chance of unnecessary failure is rather high in this protocol.

2. Reactive negotiation: the reactive negotiation is the form in which capabilities from the sender are transferred to the receiver, and the receiver decides either to accept these or responds with its own suggestions. This protocol is more likely to be successful because it allows both sides to suggest and compromise with each other. However, it may take several rounds before they can agree with each other.

Platforms can statically choose one protocol from above in respect to their own requirements in terms of the rate of success and the level of efficiency. Because the negotiation (and thereby authentication) only happens when a platform is initiated or new platforms are added, the cryptographic algorithms for each neighboring platform are static during a platform’s lifetime unless it restarts.

When this stage is successfully finished, the platform records the cryptographic options of each authenticated “adjacent” site inside the according cipher modules. If any error occurs, the security level of the link to this neighboring platform is marked “none”, showing that no secure communication service can be provided to this site.

![Fig. 5. Cooperation of protocol object, negotiator agent and SACC agent](image-url)
4.3 Implementation Algorithm

The authentication and the negotiation processes are similarly implemented by the interactions of three components in the security layer — respectively the SACC agent, a negotiator agent and a protocol object. However, different protocol objects are employed in these two processes, namely, an authentication protocol object in the authentication process and a negotiation protocol object in the negotiation process. Figure 5 illustrates how these three components interact and cooperate.

The process starts by one SACC agent sending the request message to the SACC on a remote platform. On the local platform, when receiving a positive response, the SACC generates a negotiator agent, and a protocol object which defines the steps to be followed by a negotiator agent. On the remote site, after sending a confirm message, the SACC initiates a negotiator agent and a protocol object as well. Afterwards, the process is taken over by negotiator agents on both sides, and SACC agents are then free to perform other tasks such as preparing the secure ACL message and forwarding it. Usually, the protocol objects are used in the form of a matched pair on both sides. One is labeled with the role of “SENDER” and the other with “RECEIVER”.

Once the process is finished, negotiators on both sides will report the final results to SACCs. If the result is positive, the negotiator provides the achievement as well, e.g. certificates or the shared cryptographic options and secret key. However, if any error happens during either process, failure messages are reported to the SACC indicating the reason of the failure. The error TIME_OUT occurs when no reply is received in a limited time period. The error OUT_OF_ORDER occurs when the incoming message does not match with the message that is expected. Other possible errors including the absence of specific content, verification failure (if the verification of the identity fails, then the authentication protocol fails), or unreadable messages, are all marked with the error “WRONG_MESSAGE”.

![Fig. 6. Secure Execution Environment](image-url)
Secure Execution Environment Service

In the secure execution environment service, some platform resources and some services provided by agents are protected in the security layer, which is only accessible to authorized agents. Whether an agent is eligible to access or not depends on the proof of the agent’s identity, that is, the authentication of the agents. According to our trust model the mobile agent is authenticated by the signature of its owner, who has a valid certificate issued by CA. This authorization mechanism in an agent platform provides a safe binding between the visiting agent and the local environment.

Following the basic idea of the authentication of agents and authorization, our proposal towards a secure execution environment is centered on the “permission credential”--- a proof of the agent’s permissions. Three components are involved here. A Credential Granting Center (CGC) is to authenticate the mobile agents and issue permission credentials to them. A policy Server (PS) authorizes the agent with specific access rights, and decides the major content of the permission credential. Authenticators that guard the resources or services are designed to verify the authenticity of permission credentials.

5.1 Illustration Scenario

A complete process, from the authentication of the agents to the authorization of their access, is illustrated in the figure 6. When the agent arrives or starts, agent authentication is the very first step towards platform security. This process is mandatory for all incoming mobile agents. Agent authentication is performed by the Credential Granting Center (CGC) in the security layer of the platform. Agent candidates are required to provide valid proof of their identities when under authentication, which are the digital certificates and signatures of the owner. Fake proof such as an expired certificate or a false signature may lead to the rejection of the agent’s request for a credential.

Once it regards an agent as legitimate, the CGC needs to consult with the policy server (PS) for the according authorizations that will be given to this agent. The policy server is an independent component that provides policy services such as adding, deleting and querying on policies. It manages both the platform-level and application-level policies. However only platform-level policies interest us within the scope of platform security. Access rights issued to each possible agent are defined in the format of readable platform policy items, which can be understood by the PS. The PS makes a decision according to pre-defined policy items and then returns its decision back to the CGC. On receiving the answer from the PS, a permission credential is created by the CGC, which includes all the permissions authorized to this agent, whether this agent applies for them or not. In the following steps, the agent is free to apply those resource or services authorized to it in the permission credential, which acts as the identity of the mobile agent in the current platform. If the permission credential is verified to be valid, an authenticator approves the access requirement of the agent to the resource or the service.

Since the mobile agents are only one hop in most cases, by generating the permission credential that is valid in the whole platform, the complex process of identity verification of mobile agents is performed only once, no matter how many
resources and services it may need in this platform. Instead, the verification of credentials occurs whenever the resources are needed, which consumes a significantly lower amount of computation. By checking the signature of the CGC and comparing the fingerprints taken out of the credential with the actual agent, the authenticator will decide if this credential is genuine and if the agent presenting it is the original owner. If the answers to both answers are “YES”, the authenticator will extract the piece of access right from the credential and allow the access of its guarded resources or services.

5.2 Permission Credential

A permission credential is the valid proof of an agent’s authorized access rights inside an agent platform, which consists of agent-identity-related attributes and access rights. These attributes include the name of the agent, owner of the agent, a timestamp, expire time, and most important, a hashcode of the agent as its unique “fingerprint”. The access rights are written by the CGC in terms of strings, provided by the policy server.

The permission credential in our system is based on both the public key and the secret key encryption. The digital signature of those items signed by the CGC’s private key provides the integrity and authenticity of a valid permission credential. Before being issued, credentials are encrypted by a secret key shared by the CGC and the authenticators in the current agent platform. Therefore the encryption of the entire credential permission promises that its content can be understood by the authenticators only without being detected by other entities inside the platform. Once a credential is issued, it can be used multiple times for different resources or services before it expires. Because it is encrypted by the CGC, the credential is safe to be carried by an agent without being compromised by it. Furthermore, because the secret key shared between the CGC and the authenticators is platform specific, a credential is only valid in the current platform where it is issued. If an agent tries to use its credential outside the platform, it will be simply rejected by the examining authenticator because this authenticator cannot read it properly. One example of the permission credential and how to use it is in the ACL message that is illustrated in the following figure 7.

![Permission Credential and agent’s request message](image-url)
The design of the permission credential ensures the safety in its distribution and preservation. Firstly, a permission credential cannot be stolen and used by the agents of other users. The encrypted attribute of an agent’s hashcode, as the fingerprint of an agent in the credential, ensures that only the original agent can provide the same hashcode and can therefore pass the verifications of the authenticators. Any agent who extracts a permission credential from another agent cannot pretend to be that agent. Secondly, a permission credential ensures that an agent cannot forge a credential or add more access rights. This is achieved mainly by the encryption of the credential, by the secret key shared between the CGC and the Authenticators, and the signature of CGC. Thus, even an agent with a legitimate permission credential can neither have the knowledge of this credential’s content, nor have the chance to forge it. In other words, they cannot add any unauthorized access rights by themselves.

6 Summary and Future Work

This paper has presented a security architecture based on the FIPA-OS agent platform. It addresses the security issues that exist in the message oriented agent system. Two goals are explicitly considered in our system: the protection of agents against the outside network and the protection of platforms against unauthorized access. Consequently, both the secure communication service and the secure execution environment service are described. The key point in this architecture is that we support the optional security services rather than the enforcement of security policies. In respect to threats that are not addressed, a trust model is presented in which a CA is regarded as the root of all trust relationships. The major contributions of this paper are (1) A proposal of a two-layer security architecture compliant with FIPA which provides optional security services to agents. (2) A negotiation mechanism that accepts cryptographic diversity of platforms, such as different algorithms or security levels. (3) The design of the permission credential, which is domestic proof for the agent without being misused by others.

However, there are still several interesting areas for the future work, and the first one is related to the trust model. Considering the complex situation when the revocation of credentials or certificates happens, a more flexible transmission of trust should be found instead. Secondly, as we have mentioned before, the possibility of malicious platforms make the system much more challenging. Furthermore, in this paper we consider the stationary agent and the one-hop mobile agent only. In a more advanced approach, the agents may be given the ability to move around from one site to another. The corresponding challenge derives from the difficulty of keeping the working or computation state of the agent safe. Rather than the mechanism of the mobile code signing used in our system, a new approach is needed to prove the agent’s integrity without a signature, and the tracking of the agent should be considered as well in this case.
Reference

23. S. Poslad, M. Calisti: Towards improved trust and security in FIPA agent platforms, the Autonomous Agents 2000 Workshop on Deception, Fraud and Trust in Agent Societies, Spain, 2000.
Author Index

Adler, M. ...................... 184
Agoulmine, N. ................. 235
Amin, K. A. .................... 41
Bradshaw, J. M. ............. 184
Campadello, S. ............... 73
Cardoso, A. R. ............... 163
Celestino, J. Jr .............. 163
Cherkaoui, S. ................ 11
Fonseca, M. ................... 235
Fornaciari, W. ............... 153
Gervais, M.-P ................. 219
Glitho, R. ..................... 124
Gonzalez-Valenzuela, S. ..... 63
Goutet, S. ..................... 124
Grossner, C. ................... 114
Harroud, H. ................... 114
Helvik, B. E. .................. 255
Horlait, E. .................... 1
Huang, F. Y. .................. 209
Impey, R. ..................... 277
Jamadagni, S. ................. 269
Karmouch, A. ................ 114
Karmouch, A. ................ 277
Khan, B. ....................... 103
Khoumsi, A. ................... 11
Kleiner, D. D. ................. 103
Knight, G. ..................... 139
Konaré, D. ..................... 195

Lakhdissi, M. ................ 114
Lee, D.-I. ..................... 93
Leung, V. C. M. ............. 63
Li, X. ........................ 21
Li, Y. ........................ 51
Liotta, A. ..................... 139
Mahan, M. ..................... 184
Marshall, A. .................. 235
Mayes, J. T. ................... 41
Meng, L. ........................ 51
Mikler, A. R. .................. 41
Muscetariu, F. ............... 219
Namgoong, H. .................. 93
Nayak, A. ..................... 245
Pavlou, G. ..................... 139
Perry, M. ..................... 173
Pierre, S. ...................... 124
Pierre, S. ..................... 195
Piuri, V. ....................... 153
Prestileo, A. ................... 153
Puliafito, A. ................... 31
Quintero, A. ................... 195
Roughana, N. .................. 1
Santoro, N. .................... 245
Satoh, I. ....................... 81
Skillcorn, D. B. .............. 209
Suh, Y.-H. ..................... 93
Suri, N. ....................... 184
Talmage, D. ................... 103
Tomarchio, O. ................ 31
<table>
<thead>
<tr>
<th>Author</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Velazquez, E.</td>
<td>245</td>
</tr>
<tr>
<td>Vita, L.</td>
<td>31</td>
</tr>
<tr>
<td>Vuong, S. T.</td>
<td>63</td>
</tr>
<tr>
<td>Wittner, O.</td>
<td>255</td>
</tr>
<tr>
<td>Yoo, J.-J.</td>
<td>93</td>
</tr>
<tr>
<td>Zaccaria, V.</td>
<td>153</td>
</tr>
<tr>
<td>Zhang, M.</td>
<td>277</td>
</tr>
<tr>
<td>Zhang, Q.</td>
<td>173</td>
</tr>
</tbody>
</table>