

# Bridging Semantic Gaps with Migrating Agents

Susan L. Mabry<sup>a</sup> and Lubomir F. Bic<sup>b</sup>

<sup>a</sup>Whitworth College  
Spokane, Washington  
smabry@whitworth.edu

<sup>b</sup>University of California  
Irvine, California  
bic@uci.edu

## Abstract

This paper professes how the SimAgents[1] migrating agent system bridges semantic gaps, particularly as encountered in modeling and simulation of flows. SimAgents provides an underlying execution model true to the domain view of order and structure, thereby providing greater programming ease in a distributed environment. Emphasized are autonomicity among distributed entities, parallel processing, open-endedness, intuitive decomposition and interactive simulation steering. An implemented cardiovascular model addresses semantic gaps demonstrating development flexibility and functionality.

## 1.0 Introduction

Agent-based technologies are one of the most important computing shifts in recent years. There have been many environments and languages developed to facilitate distributed and parallel processing. Ease of programming has remained a challenge in such systems. We contend that this paradigm offers solutions for bridging semantic gaps between program code and natural domain through more inherent programming approaches.

SimAgents[1] is a fully distributed, migrating agent system implemented in Java[2]. The potential of agent-based systems for modeling and simulation domains has been an application area largely overlooked. Agent functionality employed is especially suitable for modeling and simulation of domains having distributed entities, autonomous flow patterns and dynamic reactive events. The problem has characteristics and behaviors of multiple, integrated yet decentralized sub-systems with multiple controlling properties. Overlaid flow properties are often evident such as bifurcations and merges, pressure waves, and reactive flow mechanisms. This class of simulation models has previously been difficult to achieve in distributed computing. The SimAgents computational model closely corresponds to characteristics of autonomous flowing entities. A high degree of expressiveness yet simplicity results from close alignment of computing processes to actual domain behaviors. Consequently, such a system is a natural solution for this class of modeling and simulation problems.

In the paper, we first explore the philosophy of semantic gaps in program code. An overview of the SimAgents programming model is then presented. An implemented cardiovascular model reflects semantic problems

addressed by SimAgents. Different program models are compared, relative to semantic issues. Related work and performance is discussed. Finally, we conclude with how this agent framework inaugurates new and innovative ways to manage simulation complexity.

## 2.0 Semantic Gaps

Semantic gaps exist in the form of a lack of affinity to the original domain in computational order, in misrepresentation of dynamics and inaccurate contextual abstractions. Programming must always be conducted within constraints of the host architecture and of the processing paradigm, usually not aligning with the natural domain structure. Semantic gaps are encountered through the translation process of actual domain to program code. There is a disassociation from natural behavior to program code. These aspects leave the programmer with a difficult development environment and limitations in representation management of complex problems. Figure 1 illustrates characteristics encountered as a modeling process moves from natural domain to mathematical model to actual program code.

Typically, flow problems are forcefully modeled in architectures that do not assimilate the natural system. An artificial representation forces computations with the goal of deriving desired results. The order or determiners of actions often do not align with the natural behavior. Component representation is usually somewhat isolated and does not reflect interactions with other sites. Sequential for-loop structures of fixed computations and behaviors does not adequately address the semantics of complicated flow problems. At the other extreme of computing architecture, synchronously controlled mesh of parallel computations likewise do not accurately depict autonomous behaviors. Environment characteristics limit representational capabilities and result in difficulties of programming, debugging, and understanding. In order to accurately depict the natural behavior of flow problems, clearly semantic gaps must be bridged between the natural and computational systems. To bridge semantic gaps, the underlying execution model needs to be true to the domain view of order and structure.

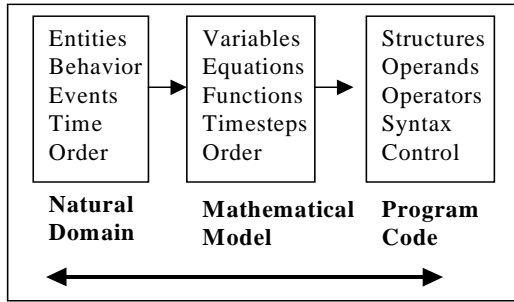


FIGURE 1. Semantic gaps between application domain structure and program code.

### 3.0 The SimAgents System

SimAgents consists of an AgentEngine and of an Application Development Framework. A flow-based topology affords natural order of control and communication. Context sensitive agents are responsive to both regional location and the current state of interaction. Powerful multi-threaded agent handlers implicitly manage agent migrations. The programmer is shielded from migration details, from distributed physical locality and from distributed communication protocols. Java structure and code concepts further encourage a programmable environment.

### 3.1 Programming Model

SimAgents is comprised of autonomous migrating *agents* and cooperative stationary *regions*. Objects representing regional entities are mapped to distributed network host sites. Regions retain local states, properties and conduct local behaviors. Agents *flow* along *paths* or *itineraries* of regions, performing actions and coordinating local behaviors. This *flowing* movement and related actions at stationary regions are analogous to entities moving through their own natural physical domains. The agent-based system can be thought of as a coordinating group of agents working in a framework of spatially oriented regions operating with their own individual agendas.

*Agents* are task oriented active objects, carrying boundary value objects and coordinating regional methods. Individual agents migrate through the logical network of regions according to their own behavior and tasks. Agents are context sensitive, detecting their physical location or an event, and acting accordingly.

*Regions* are stationary, passive objects comprised of state variables and local methods. One or more regions may reside on a physical processor host. In the partitioning scheme, regions are organized by spatial properties or by behavior roles. Local behaviors are invoked by visiting agents. Visiting agents can acquire computed values and carry them forward to other regions. Communication and state changes are transmitted by visiting agents to regions and picked up by other agents as they traverse the region, much as in the natural course of events.

A *path* is a name for a predefined series of ordered links between regions. An agent is simply fired onto a path. Because of the path abstraction, it automatically identifies properties of the path flow to travel and act upon. Paths are normally a static configuration, assigned when the system is initialized. Paths provide a means of ordered agent migration and another layer of abstraction in the form of aggregates of regions or behaviors of agents. Thus, functionality and behavior may be coupled to well-defined flows. An *itinerary* consists of an explicit designation, also ordered, of one or more regions that an agent is to traverse. An itinerary may be assigned dynamically, “on-the-fly” without regard to a territorial flow. Figure 2 depicts an abstract view of agent migration among regions. An agent is shown following an implicit path X along regions. Another agent is shown following an explicit itinerary from region A to region D. Yet another agent has been fired, concurrently, from the user interface to region B.

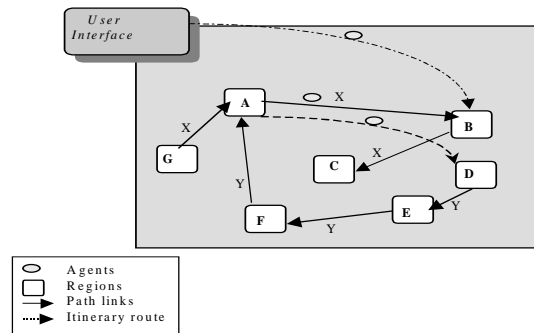


FIGURE 2. Path and Itinerary traversals of agents among framework of regions.

A logical network distinct from the physical network is established providing the perspective from which the user views. A logical network consists of regions and paths of regions, of which agents will travel. Once an initial configuration is declared, the user is aware of logical distribution of objects but not of physical mappings and need not be concerned with establishing communication channels or with tracking distributed locations. Figure 3 illustrates the separation of logical network and underlying network of host processors.

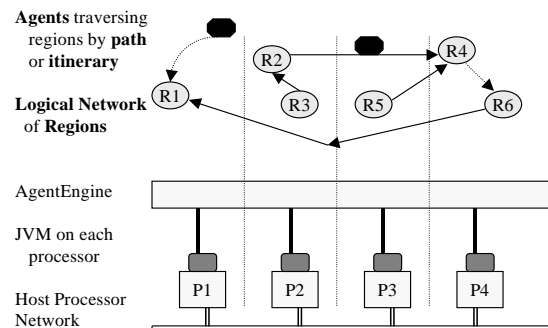


FIGURE 3. Logical Network over Network of Physical Host Processors

Dynamic agent composition is accomplished with a *fire* statement. *Fire* invokes, initializes, registers and puts into motion a new agent. When an agent is fired, it is given either a path or itinerary to follow. Although an agent follows a given path or itinerary without intervention, it is also possible to explicitly change an agent's navigation with a *jump* statement. Jump will halt the agent's current execution, capture state, migrate to the target region, restore its state and resume execution.

As in most mobile agent systems, autonomous migration provides the ability for each object to navigate through the network according to its own behavioral context. Distinctive in SimAgents is the aspect of using paths and navigation as another layer of abstraction, facilitating another dimension as a structuring instrument. Context sensitive agents perform local computations or actions relative to their perceived location and then are implicitly transported to the next region in its given path or itinerary. Navigational autonomy features support flexibility of a dynamically changeable environment. Firing of agents during runtime, without halting normal execution, is particularly effective for interactive simulations, providing two forms of simulation steering. Internal open-loop steering reacts to internal state changes and is accomplished through dynamic firing of an agent or of multiple agents upon a detected condition at a region. External event steering reacts to events introduced from a user interface.

### 3.2 AgentEngine

The AgentEngine, transparent to the user/developer, administers the runtime infrastructure. Key to the programming environment are agent and region functionalities defined in the AgentEngine. Subordinate application classes inherit all properties and methods of their respective AgentEngine superclasses. Distinctive SimAgents features enabled are automatic agent migration, dynamic simulation steering, multiple location independent registries, and territorial or itinerary navigations.

Most prominent qualities of Java used are Java Remote Method Invocation (RMI), object serialization, multi-threading, classloaders and reflection. Traditional object-oriented concepts carry heavily into SimAgents including inheritance, encapsulation and polymorphism. Important enhancements provided by SimAgents include implicit agent migration mechanisms found in the agent handlers, AgentReceiver and AgentSender. Migration occurs region by region. Classes are implemented as multithreaded objects, managing incoming and outgoing agents, synchronizing their processing, determining target destinations and handling transport. Figure 4 shows the object hierarchy for the AgentEngine as it relates to the Java Class Library and to application classes defined by the developer. The middle layer encompasses the AgentEngine, inheriting fundamental classes from standard class libraries. Application classes portray model entities, inheriting functionality from AgentEngine classes.



FIGURE 4. Object hierarchy of AgentEngine to Java Class Libraries and Application Classes

### 4.0 Map Cardiovascular Model to SimAgents

The cardiovascular system is a prime example of a class of problems for which SimAgents is designed to support. The basic cardiovascular model implemented in SimAgents has been adapted from previous work, CVSys[3]. The domain has inherently parallel and distributed physiological components with multiple asynchronous inputs. Circulatory flow is modeled as a bio-fluid, mechanical full-body cardiovascular modeling system with pulsatile flow. Nonlinearization is discretized over regional segmentations and time steps. The model supports the premise made by Karlsson[4], stressing that any effort at modeling the arterial tree should resolve characteristic features such as distributed resistance and the ability to incorporate local variations in segmental compliance. After leaving the heart, flow is pulsatile and bifurcates to smaller, tapering vessels and to distributed peripheral regions, through peripheral organ beds, then merges back into small venous vessels, continues merging into larger venous vessels until returning to the right heart. Reactive mechanisms and autoregulation further complicates the domain. There are multiple dimensions of determining factors affecting system dynamics and rates. Territorial regions play different participatory roles.

The circulatory physiology is implemented as a series of heart chambers, arterial regions, venous regions, distributed and parallel organ bed perfusions with arteriole regions and venule regions. Figure 5 depicts the decomposition of physiology into computational *regions*. Agents representing blood flows, migrate from one region to another along emanating paths analogous to natural blood flow[5].

Given the assumption that an *Artery* class has already been defined that extends the AgentEngine *Region* superclass. Upon system configuration, multiple instances of the *Artery* class would be instantiated and assigned unique names to represent the many arterial regions. A *Configuration* file maps local regions to physical processors with static links, thereby establishing a logical network.

Blood flow is converted into agent behavior. Once the agent has arrived at a region such as *aorta*, coordination statements within the the *arterial agent* class are conducted. Further migration is handled implicitly handled and subsequent coordination statements at those regions conducted. Thus, the agents flow along paths or desig-

nated itineraries, coordinating local behaviors among stationary regions. In our model, multiple types of flows are evidenced such as forward circulatory flow of blood, pressure waves, and adjustment signals such as baroreflex or cardiopulmonary responses. Flows are all programmed into migrating agents, carrying behaviors specialized for their particular actions. Simulation steering is evidenced through numerous dynamic firings of agents. Internal steering is accomplished by autonomous agents enacting reactive compensatory mechanisms over the layer of basic circulatory flow. External steering allows reactions to events introduced from a user interface, without halting the system during runtime execution. Numerous types of flows can be overlaid onto such a system because of the open-ended, extensible qualities of migrating agents.

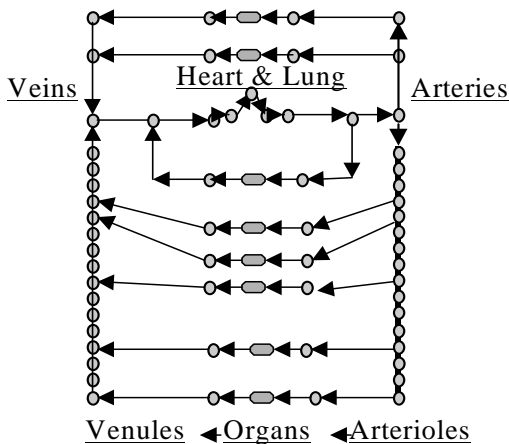


FIGURE 5. Basic closed loop circulatory physiology partitioned as stationary *Regions* in SimAgents.

In the resulting implementation, overlaid pulse waves are reflected as agents traverse region to region. Changing heart rates are transmitted via agent migration and reflected throughout the cardiac cycle. A graph example of a resultant pressure wave is shown for a specific vessel region over a given time in Figure 6.

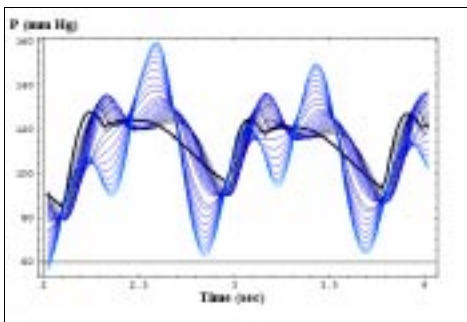


FIGURE 6. Pressure wave for vessel region over time.

## 5.0 Programming Model Comparisons

The *sequential* approach does not support distribution, concurrency and distributed controls over processing, hence it cannot preserve a natural order of events.

Although code is certainly more compact and contained cleanly within for-loop iteratives, it has considerable semantic gaps from the original application domain. With increased model complexity such limitations severely impact representative capability and management of complexity. The application is fit in a box of processing constructs rather than aligning processing to the natural order of domain events.

In comparing *message-passing* with an *agent-based SimAgents*, major areas of differences include (1) an agent-oriented approach versus a machine-oriented approach (2) concept of a logical network and (3) programmability.

**Agent-oriented versus Machine-oriented.** Most obvious differences lie in the fact that SimAgents controls and actions are *agent-oriented*, whereas message-passing is *machine-oriented*. In the agent-oriented approach, design is centered around agents visiting stationary regions, perceiving and reacting to their location. Hence actual entity roles are critical to design. Agents enact natural flows with various driving forces, having a close affinity to natural flow. In a machine-oriented approach, the computational mapping must be according to machine. The developer must even think in two levels: machine and then associated region. Obviously, because such a design must revolve around machine configurations rather than problem domain, the machine oriented approach loses the ability to bridge semantic gaps.

**Logical network.** A logical network of regions in SimAgents provides transparency from the physical network of processors. Logical regions coincide in a one-to-one basis with stationary domain regions. Because of clear alignment of regions and agents to specific computational processes and entity properties, reconfiguration is easily accomplished without affecting other region or agent class codes. There are no processor communication primitives in the SimAgents code, rather actions are collaborated with calls from the agent upon that agent's visit to a region. Message passing assumes one process mapped to one processor. It is a static arrangement with no notion of a logical network. If more than one region is to be mapped to a processor, related computation must be included in the single process executing on that physical processor. A change in configuration requires major reprogramming. There is no notion of movement, only of messaging between processors. In message-passing, all actions are coordinated by explicit send and receive communication primitives. Data is formally sent and received via messages.

**Programmability.** The open-endedness of migrating agents provides a natural solution for conditional branches carried out in autonomous flows. The branching flow is processed with a branching agent and resulting properties are carried with the flow to the receiving region. SimAgents utilizes its flow-orientation to enforce synchronization. SimAgents is easier to program largely because of clear decomposition into agents and regions, congruent

with the natural system. The concept of using flow and implicit mechanisms to reinforce order is consistent with our primary goal of providing an underlying execution model true to domain view of order and structure. In message-passing, the programmer is immersed into deciphering between processors and entity functions. Dealing with distributed communications and physical processor mappings divert attention from the domain and complicates programming drastically. Management of conditional flows is more difficult to implement. Changes necessitate significant reconfiguration. The environment is simply more difficult to conceptually envision the domain and to program the application.

## 6.0 Related Work and Performance

The mobile agent discipline has emerged from combining concepts of the artificial intelligence and distributed computing communities. Focus of these systems have primarily been as internet service providers, information retrieval and filtering, network and resource management. Most have similar purpose and present largely common functionality such as AgentSpace[6], Aglets[7], Voyager[8], Telescript[9], Mobile Object Workbench[10]. A few have somewhat more general application such as UCI-Messengers[11] and Agent-Tel[12]. SimAgents is especially designed to address the largely unexplored arena of modeling and simulations. Although many distributed and parallel environments have focused on model and simulation applications, it is our contention that functionalities found in migrating agents facilitate a dynamic, distributed, open-ended execution model, capable of processing and controlling threads analogous to the natural domain. Because the code is more closely aligned with the natural domain, programming is more inherent and extensible.

The current version of SimAgents has not been tuned for performance and suffers high overhead in the agent firing operation. However, the performance numbers are strong enough to suggest that a combination of optimizations that are currently being implemented will make SimAgents competitive in performance while retaining its design principles.

Comparisons with other distributed systems over different network speeds have been made using a simple bifurcating flow problem[1]. Figure 7 displays relative speedup demonstrated by SimAgents and MESSENGERS-C[13]<sup>3</sup>. MESSENGERS-C has been optimized with pre-compilation of individual Messengers to C and performance in both systems is limited by the degree of parallelism and data dependencies of the simple problem. Nevertheless, SimAgents demonstrated a respectable comparison given the lack of current optimizations.

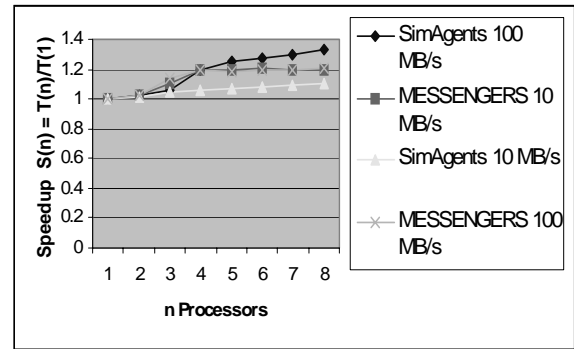


FIGURE 7. Relative Speedup of SimAgents and MESSENGERS Versions

On the full cardiovascular testbed problem, the following results in Figure 8 were demonstrated in speedup over 100 MB/s Fast Ethernet connections. In the case of a steady state circulatory model without dynamic responses, a complete heart cycle is represented by 80,000 operations. Experiments with internal open-loop reflex responses and external event entries after the initial stabilization also prove interesting, although difficult to interpret effects of the varying level of computations and processing performance.

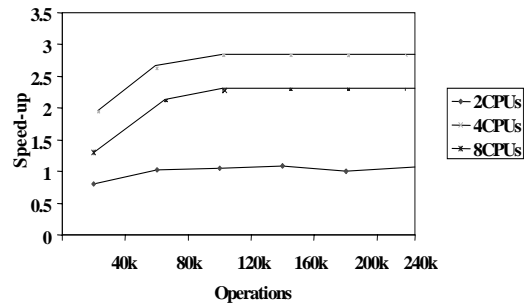


FIGURE 8. Speedup for SimAgents configuration upon 1, 2, 4, and 8 processors

Agent Migration shown in Figure 9, includes state capture, time in dispatchQueue, serialization, transport time in arrivalQueue, deserialization and restart. The fire operation is significantly more costly than basic migration, reflecting roughly a twice the overhead costs. Fire includes agent object instance creation and migration time.

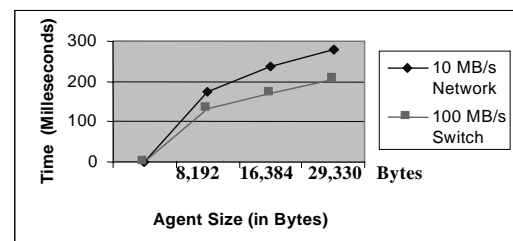


FIGURE 9. Migration time per agent size.

The proven performance of interpreted bytecodes and accomplished performance is deemed reasonable for

the designated purposes of SimAgents. While for time critical systems this approach is clearly not effective, it is significantly faster than pure interpreted approaches and with proposed optimizations, SimAgents is expected to be highly competitive in performance with other distributed systems.

## 7.0 Conclusion

In this paper, we have defined typical semantic gaps and how SimAgents addresses issues with migrating agents. It has long been recognized that cardiovascular modeling would greatly benefit from distributed, parallel processing. Clearly, sequential processing does not capture the distributed demeanor of the cardiovascular system. Parallel processing as with massively parallel processing approaches have been used to improve performance; however such approaches do not address the autonomous control flows found in such problems or allow a natural decomposition of the problem. Although distributed processing is desirable for a more natural depiction of the domain, traditional distributed approaches present both a difficult programming environment and obstacles in handling region-to-region differential equations of flow problems. These aspects are addressed through application of the agent-based model and through features of SimAgents. Modeling aspects of a cardiovascular model were directly enabled by an agent infrastructure included multiple autonomously controlled flows, distributed organ perfusion and short-term compensatory mechanisms in an interactive, asynchronous manner without halting the runtime system.

Features work together to preserve the correctness of the simulation model and thereby bridge semantic gaps. New dimensions of abstraction in a migrating-agent approach opens doors for more natural representation. Extensibility qualities include a very modular system with encapsulation, inheritance, abstraction, reusability and information hiding; which thereby provide means to control complexity, also contributing to true representation of a domain. An ease of programming allows the modeler to focus on accuracy of a model implementation. Unlike other environments, application development takes on an intuitive quality in this distributed environment because the agent-based paradigm is analogous to the natural domain order.

Through various evolutions in distributed and parallel processing, ease of programming has remained a challenge. A migrating agent framework presents a functional computational order and accurate decomposition into moving and stationary structures, resulting in narrowing of semantic gaps between the natural and virtual world. Functionality and a simple development environment act to bridge semantic gaps. Through decreasing problems of semantic gaps, we increase the usefulness of a system and create a more easily programmable distributed environment.

## References

- [1] S. Mabry. *SimAgents: Migrating Agents for Simulation Models*. PhD Dissertation, Department of Information and Computer Science. University of California, Irvine. March, 1999.
- [2] C. Horstmann and G. Cornell. *Core Java, Volumes I and II*. Sun Microsystems, Inc. Sun Microsystems Press. 1998.
- [3] S. Mabry, L. F. Bic and K. M. Baldwin. CVSys: A Coordination Framework for Dynamic and Fully Distributed Cardiovascular Modeling and Simulation. *Proceedings of Biomedical Sensing and Imaging Technologies*. 1998.
- [4] M. Karlsson. Modeling and simulation of the human arterial tree - a combined lumped-parameter and transmission line element approach. *Computer Simulations in Biomedicine*. Eds. H. Power and R. T. Hart. Computational Mechanics Publications, U.K. 1995.
- [5] S. Mabry, S. Rodriguez and J. Heffernan. Integrated Medical Analysis System (IMAS). In *Proceedings of WSC 97*. December, 1997.
- [6] A. Silva, M.M.da Silva and J. Delgado. An Overview of AgentSpace: A Next-Generation Mobile Agent System. *Proceedings of Mobile Agents Workshop*. Springer. 1998.
- [7] IBM Tokyo Research Laboratory. *The Aglets Workbench: Programming Mobile Agents in Java*, 1997.
- [8] *ObjectSpace: Voyager Core Package Technical Overview*. 1997.
- [9] J. White. General Magic, Inc. *Mobile Agents White Paper*. 1994.
- [10] M. Bursell, R. Hayton, D. Donaldson and A. Herbert. A Mobile Object Workbench. *Proceedings of Mobile Agents Workshop*. Springer. 1998.
- [11] M. Fukuda, L. F. Bic, M. Dillencourt and F. Merchant. Distributed coordination with Messengers. *Science of Computer Programming*, 31(2), 1998. Special Issue on Coordination Models, Languages, Applications.
- [12] R. S. Gray. Agent Tcl: A flexible and secure mobile-agent system. In *Proceedings of the Fourth Annual Tcl/Tk Workshop (TCL 96)*, 1996.
- [13] C. Wicke, L. Bic, M. Dillencourt and M. Fukuda. Automatic State Capture of Self-Migrating Computations in MESSENGERS. In *Proceedings of Mobile Agents. Second International Workshop, MA'98*. 1998.