A Mission-Based Multiagent System for Internet Applications

Glenn Jayaputera, Arkady Zaslavsky and Seng Loke
School of Computer Science and Software Engineering, Monash University, Melbourne, Australia
Email:gtj@csse.monash.edu.au, arkady.zaslavsky@csse.monash.edu.au, seng.loke@csse.monash.edu.au

Keywords: Agent Coordination, Task Decomposition, Mission

Abstract: Software agents have been one of the most active research areas in the last decade. As a result, new agent technologies and concepts are emerging. Mobile agent technology has been used in real life environments, such as on-line auctions, supply chain, information gatherings, etc. In most situations, mobile agents must be created and carefully crafted to work together almost from scratch. We believe that this is quite inefficient for application developers and users, and hence propose a system for generating and coordinating agents based on the notion of agent missions. The prototype system is called eHermes and its architecture and components are discussed in the paper.

1 INTRODUCTION

Software agents are gaining an enormous amount of interests from both the academic and industrial research communities and software developers. A lot of research has been conducted in this area, mainly in mobility and intelligence. Some of the areas being actively researched are: learning (Arcos and Plaza, 2001; Buffet et al., 2001; Modi and Shen, 2001), security, adaptivity (Arcos and Plaza, 2001; Chen and Sovrin, 2001), social interaction (Yolum and Singh, 2001), collaboration (Dutta and Sen, 2001), and coordination (Cabri, 2001; Cabri et al., 1997).

While much coverage has been given in the areas mentioned above, relatively little has been done in the area of run-time mobile agent coordination. Most of the coordination strategies developed so far are based on the blackboard concept (Cabri et al., 1997; Cohen et al., 1988). In this concept, a goal is decomposed into smaller tasks then routed to agents who have indicated a capability of solving them.[1] If a blackboard server does not have an agent that is capable of performing a task, it may ask other blackboard servers further along the hierarchy to solve it. We believe this concept is loose (little control), sporadic and hence inefficient because:

- there is no rigid and proper control over the task execution;
- there could be a race condition where two or more agents try to do a task posted on the board, hence duplicating the effort, and
- there could be a situation where an agent which has a task allocated to it will not perform the job as good as the other. However because of the race condition mentioned above, the weaker agent got the task first.

Further studies reveal that at the moment software developers tend to build agents for a specific purpose. For instance, MORPHEUS (Yang et al., 2001), CHAYANI (Dutta et al., 2001) and MAgNET (Dasgupta et al., 1999) were developed specifically for on-line shopping while a system like InfoSleuth (Bayardo Jr et al., 1998) was developed to perform information extraction from heterogeneous on-line information sources. Each of these systems performs different tasks, yet some of their tasks are the same or similar.

Based on those studies and findings, we believe that one could build an agent system that can be efficiently tailored to easily generate and coordinate agents for different domains and applications.

Our vision is to create a system (which we call eHermes) that takes a user-specified task and appropriate domain knowledge at run-time, and generates the appropriate agents and agent coordination strategies to carry out the task.

The rest of this paper is organized as follows; Section 2 discusses the general nature and classification of software agents. Section 3 discusses

[1] That is, a blackboard server is acting like a broker.

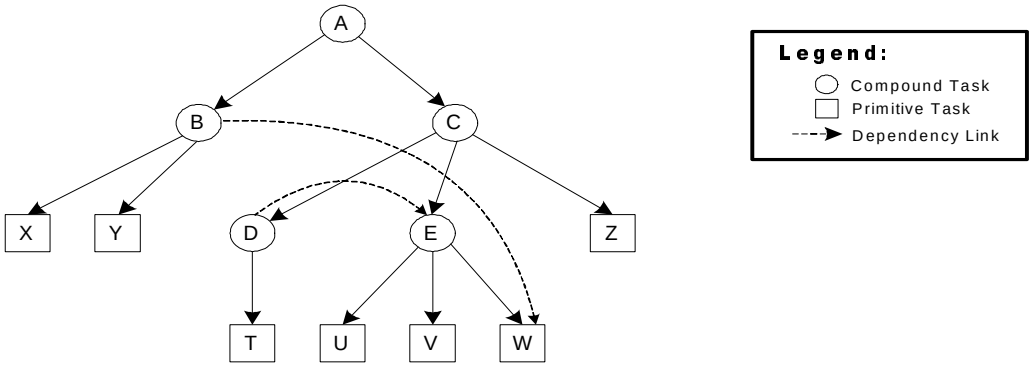

Figure1: Task Decomposition Diagram

the notion of a mission, its representation and execution, as well as the architecture of eHermes. A sample scenario is given in Section 4 along with a detailed discussion about how that scenario is represented in a mission and how eHermes executes that mission. Finally, Section 5 concludes this paper with a brief discussion about our contribution from this project and future work.

2 BACKGROUND

A software agent is typically associated with a goal or mission that it has to perform/conduct autonomously or in coordination with other agents in a multi-agent system. A mission is always domain dependent, that is, a mission is always related to a specific domain. Our study has shown that, based on how the mission is assigned to an agent and how the agent acquires the domain knowledge, the following classifications can be drawn:

- *Closed Systems*. The mission and domain knowledge are hard coded into such systems. Supplying any of this information at run-time is almost impossible. AuctionBot is one of the examples of this class. AuctionBot always performs similar actions (such as bidding) to any auction sites and its domain knowledge[2] is programmed by its developer.
- *Hybrid Systems*. In such a system, the domain knowledge is built into the agent (size overhead), however the mission is supplied at run-time.
- *Open Systems*. In such a system, both the mission and domain knowledge are supplied at run-time.

eHermes is a system that is being developed to satisfy the third class, i.e. open and yet dynamic enough so that it can be used in multiple domains (such as telecommunication and financial domains). This openness and flexibility can be achieved only if the mission can be specified and supplied to the system at run time. Furthermore, in order for the system to understand this mission, the knowledge domain in which the mission falls into must also be supplied at run time.

3 eHERMES

The main concept in e-Hermes is called the *Mission*. A Mission is defined as *a purpose or goal an agent has to accomplish to satisfy its owner's request*. A mission object can be simple (primitive) or compounded. A compound mission can be considered as a collection of simple missions.

Our mission concept extends the role notion presented by Cabri (Cabri, 2001), in that:

> *Mission embraces roles, in that roles are part of a mission. While carrying out a mission, an agent may play a number of roles.*

A user or actor supplies instructions to eHermes in the form of mission objects. These missions get interpreted and executed by eHermes by using stationary and mobile agents. In the following sections, we describe how a mission is represented and executed, and the architecture of eHermes.

[2] Such as English or Dutch Bidding system, Open or Close Bidding system, etc

3.1 Mission Representation

For an agent to autonomously perform tasks on behalf of a user, it must have a mission. Hence, a mission can be elaborated into *a set of tasks that an agent must perform*. If all the tasks specified are performed, it is concluded that it has achieved its mission.

In eHermes' mission, there are two types of tasks, namely: *compound* and *primitive* tasks. A primitive task is defined as a basic action that an agent can perform directly. A compound task, on the other hand, is regarded as a task that is composed of primitive tasks and/or other compound tasks. In order to understand this task-type definition, consider the following scenario. Let us assume that our agent is a robot that can move from one location to another. In this scenario, the ability to move robot's right and left foot forward is considered to be the basic action it can perform. In a mobile agent environment, a primitive action might be its ability to "move" from one host to another.

Tasks in a mission are inter-related to each other by dependencies and external constraints such as time and resources. A task might not be able to be performed until some other tasks are completed, that is, that task depends on the completion of other tasks. On some other occasions, a task might not be able to be completed because "it is not the right time yet" (time constraint). For instance, for an auction-bidding agent to successfully bid for as minimal price as possible, it should wait until it is near to the closing time. If the agent bids too early then other agents or user might overbid our agent[3] resulting in the final price being too expensive.

Therefore, it is identified that in order to "instruct" an agent to do something, we must provide it with a mission and that mission must be in a form that the agent can understand and hence be able to execute it.

A mission in eHermes is represented as an attributed tree structure. eHermes does not use the basic AND/OR tree because it requires a finer granularity in defining dependencies between tasks. With AND/OR tree, dependencies are limited to the sibling nodes only, and hence dependencies from any task to any task could not be captured and represented. The ability to specify tasks dependencies (from any task to any task) and to break goals into sub-goals using a tree structure is regarded as very important for a software agent to understand its mission.

We call this tree-structure a *Task Decomposition Diagram* (TDD). Our task decomposition diagram can be considered to be similar to TAEMS' (Decker, 1994) Task Tree Structure but it is simpler and adequate for our purposes.[4] Figure 1 illustrates the task decomposition diagram.

In Figure 1, circle nodes represent compound tasks while square nodes represent primitive tasks. A compound task is composed of one or more tasks and these tasks can be composed of one or more compound or primitive tasks. The root node; A is regarded as the mission itself. A is said to be achieved if and only if tasks B and C are completed. In turn, task C is completed if tasks D, E and Z are completed.

Dependencies are represented by dashed-arrows in a task decomposition diagram. Task T1 is said to be dependent on task T2 if there is a dashed arrow drawn from T1 to T2. Taking this notion into account, we can say that in Figure 1, task B depends on W and D depends on E. This means that task B cannot be executed until W has. Subsequently, task D cannot be carried out until E has been completed.

Task dependencies are important aspects that have to be captured in a mission because they represent real-world situations. For instance, a human robot that was asked to reverse a car would put the gear into reverse mode first before slowly releasing the clutch pad and pressing the accelerator pad gently. Subsequently, it will not put the gear into reverse mode until the clutch pad has been pressed.[5]

3.2 Architecture

This section presents the architecture of eHermes. The architecture of eHermes is shown in Figure 2.

eHermes is composed of two main components, they are: *Mission Generator* and *Agent Generator*. Mission generator accepts input from an actor/user and then converts it into a mission[6]. In generating a mission, the mission generator communicates with the mission repository and reuses any available sub-goals. Furthermore, valid missions that it has generated are stored in the repository for later use. Appropriate domain knowledge about the mission is loaded into the system from an external ontology repository during mission construction and

[3] It is assumed in this context that automatic bidding is not allowed

[4] Our Task Decomposition Diagram is simpler because it does not contain functions such as TAEMS' Quality Accumulation Functions.

[5] It is assumed that the robot is using a manual car in this context.

[6] Represented as a Task Decomposition Diagram.

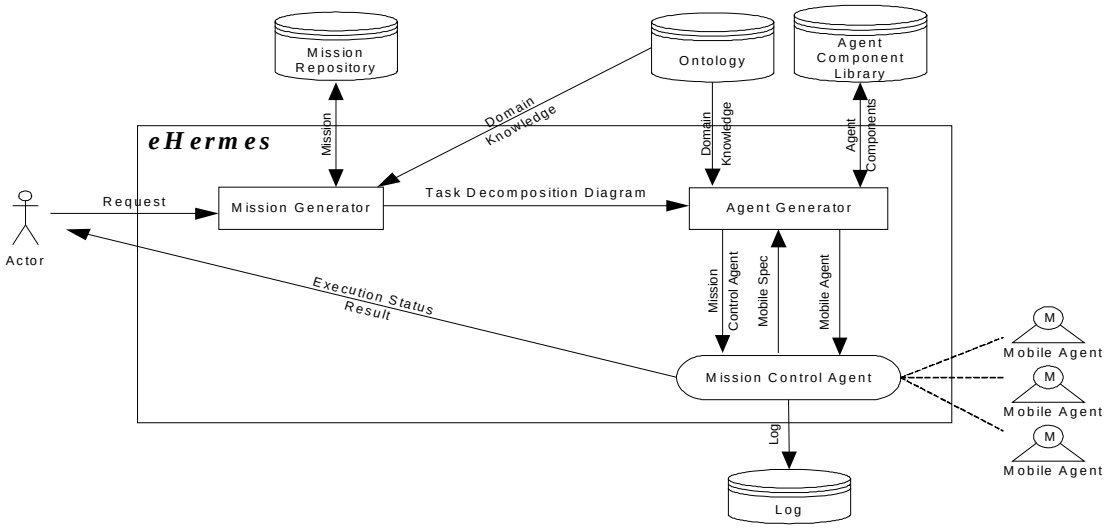

Figure2: eHermes Architecture

generation. This is to ensure that the system understands the domain of the mission it is creating.

Agent generator is a component that is in charge of creating agents within eHermes. In creating agents (either stationary or mobile), the agent generator uses the ontology repository to get appropriate domain knowledge. Furthermore, reusable agent components will be used from the Agent Component Library.

Once the task decomposition diagram (that is the mission itself) has been created, it is then passed onto the agent generator. Upon receiving this mission, the agent generator will create a special agent called *Mission Control Agent*. Mission Control Agent (MCA) is considered to be a special type of agent because it has the following properties:

- *Coordination*: it coordinates, monitors and manages the execution of a mission, ensuring the final goal is reached. If the goal cannot be completed because one or more sub-goals cannot be completed, it will then inform the actor about the situation and wait for further instruction from the actor.
- *Intelligence*: it has some built-in knowledge about how to optimize the execution of a mission by recognizing the dependencies between tasks and executes those that are not depending on each other in parallel.

The following steps show how a request from an actor will be processed by eHermes:

- The actor sends a request to eHermes. The request contains some information about the domain knowledge of this request.
- The mission generator takes this request and loads the appropriate domain knowledge from the ontology repository.
- The mission generator converts this request into a mission. During this process the mission generator will query the mission repository and reuse any existing mission.
- The mission generator sends the mission to the agent generator after the mission has been verified that it does not contain a cyclic dependency.
- The agent generator takes the mission and then builds a mission control agent for this mission and finally launches this agent.
- The MCA examines the mission assigned to it. It scans the task decomposition diagram in search for the most suitable task(s) to be executed.
- For each of the most suitable task(s), MCA creates a specification and sends this specification to the agent generator.
- The agent generator accepts the specification and creates a mobile agent. This mobile agent will be equipped with the itinerary of places it must visit. This mobile agent will be returned back to the MCA, which has requested it.
- The MCA launches, monitors and coordinates those mobile agents. Results returned by those agents will be filtered

(duplicate results are removed if necessary) and finally returned to the actor.

3.3 Mission Execution

MCA is a stationary agent, and through its whole life cycle has a managerial role. It only coordinates and oversees the execution of a task given to it. If necessary, when a mission cannot be completed due to an undo-able task(s), MCA will report to the actor about the situation. Depending on the actor's response, MCA may continue the mission, abort or let the actor redefine the mission.

A mission gets executed by first executing those tasks that can be performed directly.

Those tasks are primitive tasks. However, MCA cannot just blindly execute those tasks because there may or may not exist some dependencies between tasks within the mission. Hence, some intelligence is needed to carry out the mission.

MCA has some intelligence through its ability to:

- *Sort out which tasks need to be completed first*. Therefore, rather than just executing primitive tasks first, and then work up the tree to the root node, MCA will use dependency links to calculate and determine which tasks have to be performed first. This is because MCA understands that there is no point executing all the primitive tasks if one of them is so crucial to the mission that failing to complete this task means the whole mission cannot be completed.

  Consider the mission illustrated in Figure 1, MCA will execute tasks W first, since it knows that there is no point executing X, Y, U, V, T and Z at the same time because being unable to complete task W means being unable to complete the whole mission. Therefore, rather than wasting system resources as it would be if it executed those primitive tasks unselectively, MCA will only execute task W first. If the result of this execution is successful then it will continue with its tasks execution.

- *Execute independent tasks in parallel and hence achieve a higher throughput.*[7] For instance, taking the mission which is depicted in Figure 1, once task W has been completed, MCA will execute tasks X, Y, U and V in parallel as opposed to serially.[8] Note that, MCA does not execute task T because, since task D depends on E which in turns depend on tasks U and V then there is no point executing task T until tasks U, V and E are completed. The same explanation can be given to why MCA does not execute task Z at this point.

As mentioned previously, MCA is a stationary agent, that is, it does not change places. Therefore, in order to execute a mission; MCA needs worker agents to perform tasks. These worker agents, which normally are mobile agents, have to be created on the fly and on a mission-to-mission basis. MCA does not have the ability to create such agents and hence relies on the agent generator to do so for it.

To get the agent generator to create the agents it needs, MCA must supply a specification about the agents it wants to the generator. This specification includes an itinerary for the created agent (if necessary), task to do once it has reached the destination and how to send the result back to MCA. For the itinerary specification, the ITAG Scripting Language developed in (Loke et al., 2001) is used.

Once those mobile agents are created and given to MCA, it can then launch them to do their tasks. MCA will wait and monitor the results sent back by those agents. Results are combined and filtered if necessary. This process is continued until the mission is completed and reported back to the actor.

4 SAMPLE SCENARIO

In this section, we illustrate a scenario where the user uses eHermes to find a movie to watch. Several assumptions are made for simplicity of this discussion, they are:

- The user's profile is available and stored somewhere in the system. This profile contains information such as user's name, address, kind of movie the user likes, the most favorable actors, etc.
- Ontologies are available and available for eHermes to use.
- There exists two contactable databases; one holds the list of cinemas in the country and the other list all currently screening movies.

[7] Assumption is made at this point that all the tasks will take a uniform time to execute.

[8] It is assumed here that there is no time constraints involve in this mission.

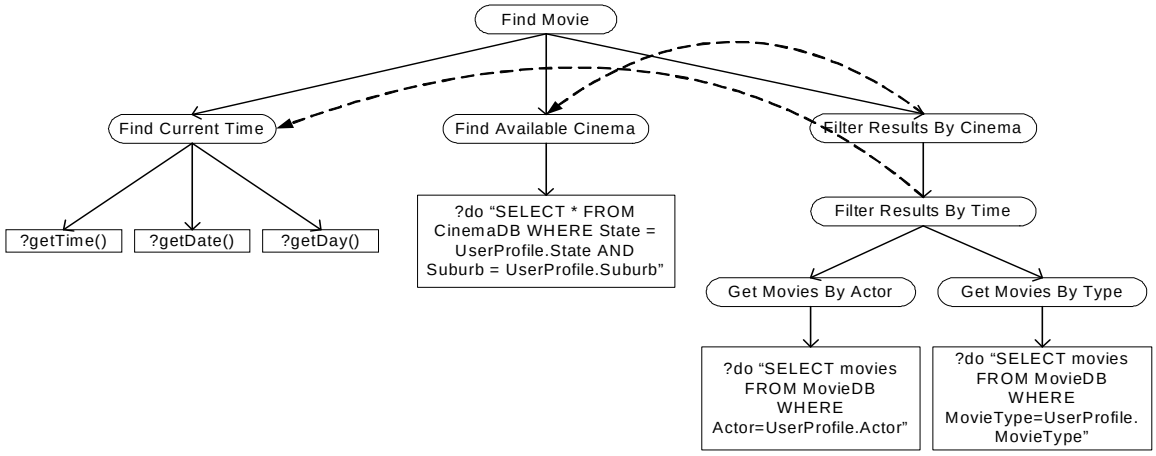

Figure 3: TDD for Finding Movie

These two databases are: CinemaDB and MovieDB.

Upon logging into the system, that user's profile is loaded into the system. From this point, the user must select the most appropriate ontology for the system to use. This is the mechanism to contextualize eHermes to a domain. Loading ontology into the system means that the domain knowledge and model are loaded, including the data model for movies and known vocabularies. For instance, the meaning of the word "surfing" is different when the context is about Internet and not sports. Hence, it is important for the system to understand the meaning of words.

Suppose that user wants to know if there is any movie he/she can see tonight. Then this query will be transferred into a mission illustrated in Figure 3.[9]

The mission generator performs cyclic dependency checking to ensuring that the mission does not deadlock. Once this TDD is passed onto the agent generator, a mission control agent is created, and once this agent comes to life it will start examining the TDD.

After the examination, MCA finds that it can executes tasks ?getTime(), ?getDate(), ?getDay(), and ?do "SELECT * FROM CinamaDB WHERE State-UserProfile.State AND Suburb=UserProfile.Suburb". Again, as discussed in the previous section, MCA is intelligent enough not to execute task ?do "SELECT movies FROM MovieDB WHERE Actor=UserProfile.Actor" and ?do "SELECT movies FROM MovieDB WHERE MovieType=UserProfile" since their parent tasks; Filter Results By Time and Filter Results By Cinema depends on Find Current Time and Find Available Cinema.

In order to execute those primitive tasks, MCA with the help from the agent generator launches four mobile agents, three of which to get current time, date and day, and the last one to get a list of cinemas that are close to where the user lives from CinemaDB. The results obtained from these actions are then stored into internal structures for later use.

Upon completion of these actions, MCA will then execute the next two tasks, they are: ?do "SELECT movies FROM MovieDB WHERE Actor=UserProfile.Actor" and ?do "SELECT movies FROM MovieDB WHERE MovieType=UserProfile.MovieType". These tasks are to get lists of movies that the user likes because his favorite actor stars in that movie and the type of movie matches his profile (e.g. action movie).

The results will then be filtered by the tasks: Filter Results By Time and Filter Results By Cinema. Once the filtering process has been completed, the result is formatted and sent back to the user.

5 CONCLUSION AND FUTURE WORK

Agent, and in particular mobile agents are created in an effort to build a distributed application. Mobile agents are often required for various reasons such as, performance issues, remote information gathering, etc. Current agent platforms offer a wide

[9] This figure uses rounded rectangle to represent compound task. This is in order for us to be able to put text inside the enclosure. While the shape is different no semantic had been changed in any way. The '?' symbol used in every primitive tasks are for programmatic notation only, they convey no semantic meaning at all.

range of services to agent developers, including mobility. However, mobility of agents is normally limited to the same host where they were launched or to other hosts that run the same environments. In other words, it is often restrained to a homogeneous environment. We believe that cross-platform migration (heterogeneous environments) is needed in order to make mobile agent technology widely accepted in the real world. However, since inter-platform agent migration is not our focus, readers who are interested in such issues should refer to the discussion presented in (Brazier et al., 2002).

We have presented in this paper an approach to semi-automatically build software agent applications. Openness and genericity notions are achieved by allowing the actor/user to specify a mission at run-time and to load appropriate domain knowledge (through ontology) during mission construction and generation.

We have also presented the architecture of our prototype system called eHermes and a representation for missions. A discussion of mission execution is also presented along with a technique to optimize the mission execution. A novelty in our approach is the notion of mission, which is used not only for identifying agents that need to be generated but also for coordinating these agents at run-time.

Our approach potentially provides increased flexibility by generating agents to accomplish a given mission, as oppose to giving a mission to a fixed set of agents already created and running whose coordination is already preset (perhaps independently to their mission). Moreover, we contend that our approach will decrease development time for multiagent systems, and with appropriate domain knowledge and components already present in the system, even non-computer scientists can utilize the system to generate agent-based applications for their tasks.[10]

Future work for this project falls into several categories: enhancements on the optimization logic and algorithms including sub-goal slicing for parallel execution, task dependency checking, and mobile agent coordination. A further aspect of our work relates to agents creating and dispatching other agents; this is non-trivial when an agent needs to have some understanding (and some ways to represent this understanding) of the agents it creates in order to manage them effectively and to reason about them. We are currently implementing a prototype of eHermes based on the Grasshopper agent toolkit (Magedanz and Bäumer, 1999).

[10] One should not need a PhD to realize the agents one can conceive in their minds, at least, not always.

ACKNOWLEDGEMENTS

Support for this project from the Australian Research Council (ARC) Linkage Grant LPO211384 is thankfully acknowledged.

REFERENCES

Arcos, J. L., and Plaza, E., (2001), Exploiting Context Awareness in Information Agents, in: *Proceedings of the Fifth International Conference on Autonomous Agents*, ACM Press, Montreal, Quebec, Canada, pp. 116-117.

Bayardo Jr, R. J., Bohrer, W., Brice, R., Cichocki, A., Fowler, J., Helal, A., Kashyap, V., Ksiezyk, T., Martin, G., Nodine, M., Rashid, M., Rusinkiewicz, M., Shea, R., Unnikrishnan, C., Unruh, A., and Woelk, D., (1998), InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic environments, in: *Reading in Agents* (M. N. Huhns, and M. P. Singh, eds.), Morgan Kaufmann Publishers, Inc., San Francisco, pp. 205-216.

Brazier, F. M. T., Overeinder, B. J., Van Steen, M., and Wijngaards, N. J. E., (2002), Agent Factory: Generative Migration of Mobile AGents in Heterogeneous Environments, in: *Proceedings of The 2002 ACM Symposium on Applied Computing (SAC 2002)*, ACM Publisher, Madrid, Spain, pp. 101-106.

Buffet, O., Dutech, A., and Charpillet, F., (2001), Incremental Reinforcement Learning for Designing Multi-Agent Systems, in: *Proceedings of the Fifth International Conference on Autonomous Agents*, ACM Press, Montreal, Quebec, Canada, pp. 31-32.

Cabri, G., (2001), Role-based Infrastructures for Agents, *The 8th IEEE Workshop on Future Trends of Distributed Computing Systems*.

Cabri, G., Leonardi, L., and Zambonelli, F., (1997), Coordination in Mobile Agent Applications, Universitâ di Modena.

Chen, H., and Sovrin, T., (2001), Steps Towards Creating a Context-Aware Software Agent System, http://www.hpl.hp.com/techreports/2001/HPL-2001-231.pdf.

Cohen, P. R., Cheyer, A., Wang, M., and Baeg, S. C., (1998), An Open Agent Architecture, in: *Reading in Agents* (M. N. Huhns, and M. P. Singh, eds.), Morgan Kaufmann Publishers, Inc, pp. 197-204.

Dasgupta, P., Narasimhan, N., Moser, L. E., and Melliar-Smith, P. M., (1999), MAgNET: Mobile Agents for Networked Electronic Trading, *IEEE Transactions on Knowledge and Data Engineering, Special Issue on Web Applications*.

Decker, K. S., (1994), TÆMS: A Framework for Environment Centered Analysis & Design of Coordination Mechanisms, in: *Foundations of Distributed Artificial Intelligence* (G. O'Hare, and N. R. Jennings, eds.), Wiley Inter-Science.

Dutta, P. S., Debnath, S., and Sen, S., (2001), A Shopper's Assistant, in: *Proceedings of the Fifth International Conference on Autonomous Agents*, ACM Press, Montreal, Quebec, Canada, pp. 59-60.

Dutta, P. S., and Sen, S., (2001), Identifying Partners and Sustenance of Stable, Effective Coalitions, in: *Proceedings of the Fifth International Conference on Autonomous Agents*, ACM Press, Montreal, Quebec, Canada, pp. 23-24.

Loke, S. W., Zaslavsky, A., Yap, B., and Fonseka, J., (2001), Scripting Mobile Agents to Support Cooperative Work in the 21st Century, in: *Proceedings of the Workshop on Agent-Supported Cooperative Work at Autonomous Agents 2001 (ASCW-2001)* (B. Lee, and Y. Ye, eds.), Montreal, Canada, pp. 1-10.

Magedanz, T., and Bäumer, C., (1999), Grasshopper - A Universal Agent Platform Based on OMG MASIF and FIPA Standards, http://www.cordis.lu/infowin/acts/analysys/products/thematic/agents/ch4/ch4.htm.

Modi, P. J., and Shen, W. M., (2001), Collaborative Multiagent Learning for Classification Tasks, in: *Proceedings of the Fifth International Conference on Autonomous Agents*, ACM Press, Montreal, Quebec, Canada, pp. 37-38.

Yang, J., Seo, H., and Choi, J., (2001), MORPHEUS: A More Scalable Comparison-Shopping Agent, in: *Proceedings of the Fifth International Conference on Autonomous Agents*, ACM Press, Montreal, Quebec, Canada, pp. 63-64.

Yolum, P., and Singh, M. P., (2001), Designing and Executing Protocols Using The Event Calculus, in: *Proceedings of the Fifth International Conference on Autonomous Agents*, ACM Press, Montreal, Quebec, Canada, pp. 27-28.