A FACTORY-BASED APPROACH TO SUPPORT E-COMMERCE AGENT FABRICATION

Steven Guan, Fangming Zhu and Min-Thein Maung
Department of Electrical and Computer Engineering
National University of Singapore
10 Kent Ridge Crescent, Singapore 119260

Abstract

With the development of Internet computing and software agent technologies, agent-based e-commerce is emerging. How to create agents for e-commerce applications has become an important issue along the way to success. We propose a factory-based approach to support agent fabrication in e-commerce and elaborate a design based on the SAFER (Secure Agent Fabrication, Evolution & Roaming) framework. The details of agent fabrication, modular agent structure, agent life cycle, as well as advantages of agent fabrication are presented. Product-brokering agent is employed as a practical agent type to demonstrate our design and Java-based implementation.

Keywords: agent-based e-commerce, agent fabrication, software agents, agent framework

1. INTRODUCTION

With the dramatic growth of the Internet, electronic commerce (e-commerce) has boomed rapidly. However, there also exist some obstacles to the success of e-commerce. Firstly, buyers may lose their way in the ocean of items available, and likely to miss the best deal. Secondly, it is a tedious task to search for a specific product through the Internet. Thirdly, some tasks, such as negotiation with multiple terms, are so complicated that they are difficult to be dealt with under the current infrastructure.

To solve these problems, agent-based e-commerce has thus become a promising technology, within which software agents play a central role [6, 14]. Software agents differ from traditional software in that they possess the properties of pro-activeness, autonomy, and mobility [1, 21]. Therefore, they can carry out delegated tasks in simple, intelligent, and independent manners. They have demonstrated tremendous potential in conducting various tasks in e-commerce applications, such as comparison-shopping [7], negotiation [14, 17], payment, etc. [10] proposes an XML framework for agent-based e-commerce, with which software agents can easily interpret XML-based documents and messages. The emergence of the semantic web [15, 19] has also facilitated agent's access to the Web.

Many agent-based e-commerce applications have emerged in recent years. AuctionBot [27] is a generic auction server that allows users to auction products by employing agents. Agents are created by sellers and buyers using the interfaces offered, and these agents can conduct negotiation with customized bidding strategies. MIT Media Lab's

Kasbah [2] is an online marketplace where buyer and seller agents can interact. Users can create buyer or seller agents, provide them with a set of criteria and dispatch them into the marketplace. Buyer agents may filter the available offers according to users' criteria, and then proceed to negotiate a deal. ICOMA [20] is an open infrastructure to simulate intelligent agent-based e-commerce, mainly dealing with product searching and filtering. However, agents in these applications operate at a single server site, and they cannot roam from server to server. Users cannot easily embed individual preferences in their agents, and are given few or no options to customize agents.

As a matter of fact, many issues have to be dealt with before this agent-based approach can be accepted widely as a new paradigm for e-commerce. Some literature has attempted to address issues such as authorization, authentication, traceability, integrity, and security [4, 11, 22, 25]. However, little effort has been devoted into the construction of agents. Some researchers provide agent development frameworks and toolkits, such as AgentBuilder [24], Zeus [3], and Aglet [18] to help users creating agents. However, such frameworks and toolkits only provide basic development environment and tools to help users create generic agents. For various agents in e-commerce applications, users still need to program specific function modules by themselves. But most e-commerce users do not have such programming skills, and the usage of toolkits may already be difficult for them. Furthermore, e-commerce agents created with different toolkits also lead to lack of interoperability, which may result in disorder and cause difficulty in communication among agents.

In order to alleviate the above concerns, a factory-based approach has been proposed for agent fabrication and integrated into the SAFER architecture (Secure Agent Fabrication, Evolution & Roaming) [29]. The objective of our scheme is to provide a convenient and safe approach to create agents for various agent-based e-commerce applications. The key point is that new agents should be fabricated by authorized agent factories according to prescribed formalities and customizations from agent owners. With our scheme, users can be alleviated from the laborious work of programming and hosts can be relieved from the risks of accommodating unauthorized agents. Agents thus created would have a common structure, facilitating communication and collaboration among them. We employ the product-brokering agent in this paper to elaborate our design and implementation. The function of a brokering agent is to accept queries from a user, visit relevant product servers to find related product information, and finally present the information to the user.

Although the concept of software factory has been studied well in the software engineering arena [5, 8], it is especially applicable for agent-based e-commerce due to the reasons explained above. To distinguish our work from the existing approaches in software engineering, we place emphasis on the fabrication procedures for e-commerce roaming agents and highlight their applications in e-commerce systems.

The remainder of this paper is organized as follows. In section 2, an overview of the SAFER framework is presented and the role and functionality of each entity in the framework are briefly introduced. Section 3 elaborates details on the agent fabrication scheme including agent modular structure, agent life cycle, and fabrication formalities. Section 4 and 5 present implementation and discussions, and section 6 concludes the paper.

2. SAFER FRAMEWORK

SAFER is a framework designed to serve agents in e-commerce and establish necessary mechanisms to manipulate them [29]. The main objective of SAFER is to construct an open, secure and evolutionary agent system for agent-based e-commerce, incorporating agent fabrication [13], evolution [28], and roaming [12]. Agent fabrication is one of the fundamental parts in the SAFER Architecture.

As shown in Figure 1, community is the basic unit under the SAEER framework. With the basic structure of communities, agents may be regulated in a tidy order and perform their tasks more efficiently. The community A in Figure 1 shows typical components in one community, including agent factory, community administration center, agent owner, product server, etc. In a real implementation, some communities may have more or less entities. For instance, several communities may share one agent factory, and there can be more than one marketplace in one community.

Agent factory is the kernel of SAFER, as it undertakes the primary task of creating agents. An agent factory only serves registered owners. Before it accepts requests from an owner, it checks the identity of the owner with the community administration center to ensure that the owner has already registered. The agent factory then fabricates new agents according to the customizations from the owner, complying with prescribed procedures. The detailed fabrication formalities will be presented in Section 3.4.

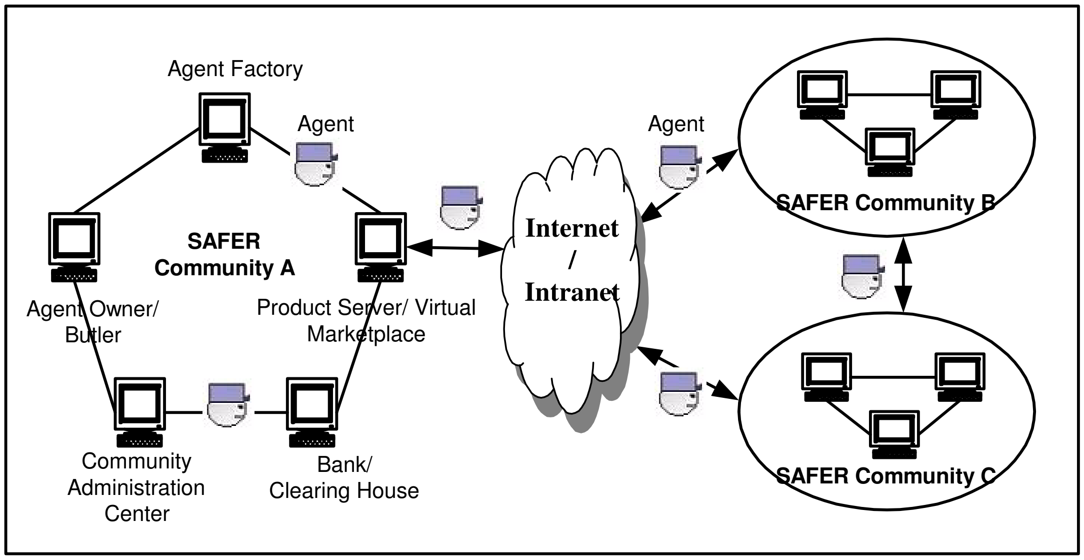


Figure 1. SAFER Framework

Community administration center (CAC) is responsible for administrative matters in a community, coordinating and facilitating activities of the entities in the community. It aims to ensure smooth running of routine operations and security of the whole community. A center has a roster of its community, which includes basic information about the entities in the community. Whenever an agent is created or decomposed, certain information will be reflected in the roster. In addition, the center does a thorough routine examination on all entities in the community and updates its roster periodically. Therefore, the center is well aware of its legal residents so that the community is well-guarded from intruders.

The agent owner stands at the top of the SAFER framework's hierarchy, since he holds the priority and responsibility for all his agents. An agent owner can act as a buyer, seller, or proxy, and he controls his agents from creation to termination. There are many agent owners existing in one community, and each owner can possess several agents. To relieve his burden, an owner can authorize an agent butler to handle most of his tasks. An agent butler assists its owner in various tasks such as authorization, payment control, activities tracking, etc. In the absence of the owner, an agent butler will, depending on the authorization given, make decisions on behalf of its owner.

Agents play an active role under SAFER. Other entities in the SAFER framework serve agents in one way or another. Each agent has a unique identity and belongs to one owner. According to tasks assigned by their owners, agents can be classified into many categories, such as negotiation agents, payment agents, brokering agents, and so on. The location of agents can change frequently. Agents can be awaiting new instructions in the owner’s computer when they are idle, or roaming from one host to another, or operating in a foreign host. An agent can learn about the owner’s preferences, evolve its ability of reasoning and adjust its behavior according to tasks assigned and resources available.

Product server and virtual marketplace are included under SAFER as typical commercial facilities to serve software agents. Product server is a service provider that mainly provides product information for agents. In a virtual marketplace, agents from buyers and sellers can undertake various tasks, such as information-gathering, advertising, negotiation, etc. For example, agents from sellers can post their product information in the marketplace, while agents from buyers can collect them. Moreover, they can communicate with each other directly in the marketplace. The marketplace can also adopt the auction business model [26]. Agents take part in some auction process on behalf of their owners. Since they have their own strategies for auction, they can place bids according to authorization, preferences, and current situation. If an agent becomes the winning bidder at last, it may continue with the process of transaction, even bring the auction targets (e.g. software products) back to its owner.

In order to facilitate agent-based financial transaction and clearance, clearing house/bank are included as separate entities in each SAFER community. The detailed payment schemes in SAFER are currently under active research [16], which is not the focus of this paper.

3. AGENT FABRICATION

There are many reasons and advantages to adopt agent fabrication. We emphasize them again as below.

- Since most users have no ability to create agents by themselves, it will be more convenient and ideal if an agent can be fabricated according to customizations from agent owners.
- Agents implemented individually can lead to lack of interoperability. This may cause difficulty in communication among agents.
- Agents created from authorized factories will generally be more trustworthy.

- It can enhance security mechanisms of agent-based e-commerce, as agents can be administered more efficiently and safely.

3.1 Agent Factory

The main task of the agent factory is to create new agents on behalf of community's members. The agent factory provides interfaces for agent owners to customize their new agents to match their requirements. As the agent fabrication services are provided only to the community's members, the agent owner needs to be a legal member of the community in order to request for the creation of new agents. (The details of agent fabrication routine will be discussed in the later implementation section.) The agent factory also maintains a database named Archive, where the records of successfully fabricated agents are stored. In addition, the agent factory also assumes the responsibilities of checking and fixing agents. The agent factory will update the Archive whenever it performs a service for the agents.

3.2 Agent Modular Structure

In order to facilitate agent fabrication, modularization is adopted to support the fabrication process. Various modules are maintained in agent factories. During fabrication, necessary modules can be assembled according to the requests from the user and guidelines for fabrication. As a matter of fact, the weight (size) of an agent is essential for its efficiency. A heavy agent with redundant modules will be less efficient, because a lot of time is wasted on transferring its fat body during roaming. Moreover, the risk of being attacked will be higher. On the other hand, if an agent lacks the necessary modules, it cannot fulfill even the basic functions. So we need to strike a good balance.

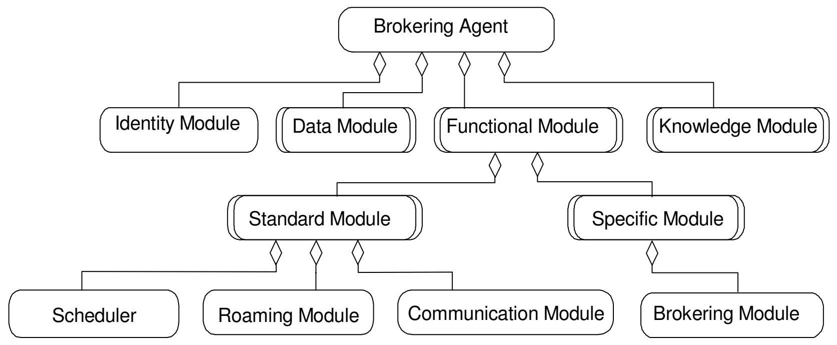


Figure 2. Agent Modular Structure

Figure 2 shows the structure of a brokering agent as well as the modules included inside. The identity module contains basic elements of the identity of an agent, such as agent ID, certificate, timestamp, etc. The data modules store information collected from hosts and logs of the agent activities. The knowledge modules store agent knowledge to support analysis and decision-making process. The functional modules provide functions that an agent may need when performing specific tasks such as communicating with other agents, roaming to other hosts, negotiating with vendor agents, brokering product information across the Internet, and so on. The functional modules comprise standard and specific modules. These modules are basic components shaping up the characteristics of an agent and a group of them can be assembled to fulfill expected functionalities. There are two choices for a standard module, i.e., direct module implementation or virtual module with Global ID (GID). GID is a string representing a standard module in agent factories. It can replace a real implementation to decrease the weight of an agent. Whenever an agent visiting a host needs to make use of standard functions that are GIDrepresented, the host will simply load the module implementations from a local database if there is one. Even if the real implementation associated with a certain GID cannot be found in the database, the host can download it from agent factories.

In most situations, agents must cooperate or interact with others to accomplish their delegated tasks, whether for individual or common goals. The communication module is the key requirement to cooperation and interaction. In order to properly understand the meaning of messages from others agents, both agents must have the same definitions for symbols (constants) used in the messages. The communication module defines the standard message structure. A message is comprised with a message type, for describing the intention, and a message body, which includes the information needed for accomplishing that intention.

3.3 Scheduler and Task Queue

As the scheduler is a key component of an agent, we discuss it in a separate section. The first responsibility of the scheduler is to map the user's delegated tasks into the agent's tasks. This means, even though the user assigns only one task, an agent may have to construct a couple of agent tasks to accomplish it. Agent tasks are maintained in a task queue, which uses a FIFO (First In First Out) queue mechanism to guarantee an orderly processing from the first task to the last. The task queue maintains two pointers, one pointing to the front of the queue and the other pointing to the end. The details will be further discussed in the implementation section.

Agent tasks can be classified into two types, namely primary task and secondary task, based on their dependency. A primary task is independent of the other tasks inside the task queue, and can be executed under any conditions. On the contrary, a secondary task depends on a particular primary task and will not be executed if its associated primary task fails. For example, the task for roaming to a product server is a primary task and an agent will execute that task by all means, while an enquiry task is a secondary one and will be executed only if the agent has successfully roamed to the destination. There is no meaning for executing the enquiry task when the agent fails in roaming to the destination, as the enquiry task is supposed to be run at that specific destination. The scheduler will therefore remove the dependent secondary tasks from the task queue, if a primary task fails.

The scheduler is not only responsible for planning the tasks ahead of time but also responsible for the following tasks during an agent's runtime:

- Make sure that the agent continues executing the correct tasks at the right host until the task queue is empty.
- Ensure that the agent stops its execution at the local host when the agent is about to roam to another host.
- Record the task queue states so that the agent can resume its execution from where it is left off at the previous host.

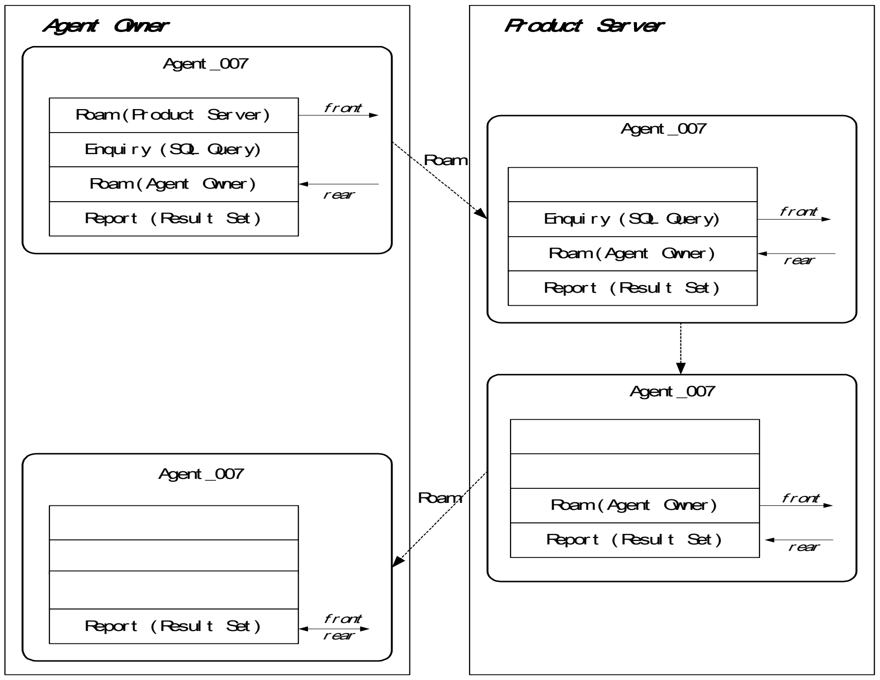


Figure 3. Agent’s Task Queue at Different Hosts

Figure 3 shows an illustration of a brokering agent’s task queue at different hosts when performing its assignment. Initially there are four tasks inside the task queue, namely, roaming to a product server, finding out product information, roaming back, and reporting the result set to its owner. The agent will pack itself and roam to the product server. If the roaming task fails, the scheduler will remove its dependent task(s), i.e., the enquiry task. As shown in the figure, when the agent reaches the product server, the task queue has only three tasks left, since the scheduler has already assigned the first task and finished it. As soon as a separate thread has been assigned for the agent, it will resume its outstanding tasks at the product server. The agent submits its queries to the agent coordinator of the product server, and may get some results back. (The detailed process will be presented in the implementation section). After the agent roams back to its owner host, it will execute the final outstanding task, which is to present the brokering results to the agent owner.

3.4 Agent Life Cycle

In order to control the transition of agent's states, an agent life cycle is constructed. As shown in Figure 4, an agent's life cycle includes six states, namely, new, ready, running, roamed, sleep and dead.

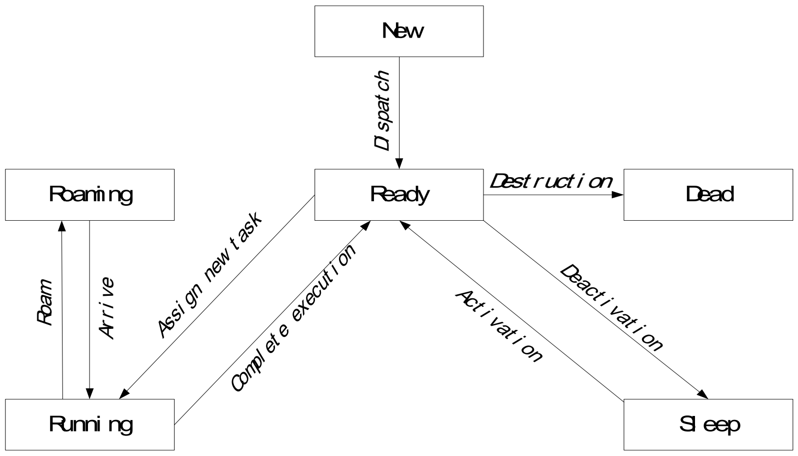


Figure 4. Agent's Life Cycle

The *new* state is the starting point of an agent's life cycle. The agent will be in the *new* state after an agent factory successfully fabricated it. The agent owner where the newborn agent is dispatched to will place it in the *ready* state, which means it is ready to undertake new assignments. When the agent owner assigns it a new task and instantiates a new agent runtime environment, the agent will move from the *ready* state to the *running* state. In the *running* state, the agent may be assigned with a separate thread for execution. When the agent finishes its assignment, it will return to the *ready* state. During runtime, the agent might roam and mark itself as *roaming*, and it will move back to the *running* state when the agent arrives at a new host and resumes its task. In order to free up resources, the agent owner may deactivate the agent, serialize it into byte streams and store it in the disk. Thus the agent will be put into the *sleep* state. The agent owner can however activate the sleeping agent and put it back in the *ready* state whenever needed. The *Dead* state represents that the agent has been destroyed permanently. An agent owner may destroy an agent whenever he doesn't need its services anymore.

3.5 Agent Fabrication Formalities

The agent fabrication process comprises of three stages: namely identification, customization, and fabrication. Figure 5 shows the whole process and message exchange during fabrication.

The *identification* stage mostly deals with checking the identity of an agent owner to make sure that the requesting agent owner is a legal member of the community. In the *customization* stage, the agent owner customizes his new agent through the interface provided by an agent factory. And finally in the *fabrication* stage, a new agent is fabricated by the agent factory based on the agent owner's customizations. In order to fabricate a new agent successfully, the fabrication procedure must pass through all three stages successfully. Some accidents may occur unexpectedly. For instance, the agent owner and the factory may not reach an agreement in the customization stage, or the agent owner has not registered yet, or messages are lost during transfer. These will result in termination of the fabrication procedure midways.

The agent owner initiates the fabrication process by sending a *Request* message to an authorized agent factory. The message contains the identification and certificate of the agent owner. When the agent factory gets this message, it checks the identity of the agent owner with the CAC by sending over a *Check_ID* message that contains the information of the agent owner. Then, the CAC looks up in the roster where the basic information of registered components is stored. Agent owners can register with the CAC at any time as long as they meet the membership's criteria of the agent community. After the CAC makes sure that the agent owner is registered, it will send back a *Confirmation* message to the agent factory. Then the agent factory informs the owner of the approval of his fabrication request with an *Approval* message. The message will contain the available agent types for fabrication. The only condition required for passing this stage successfully is that the agent owner is a legal member of the community and has registered in the CAC.

The agent owner can freely choose any agent type that is suitable for performing the intended task, and send a *Customization* message to the agent factory to indicate the selected agent type. As soon as the agent factory gets the *Customization* message, the agent factory will return a *Choice* message to the agent owner, which contains all the modules available for the requested agent type. After the agent owner customizes these modules, a *Fabrication* message will be sent to the agent factory and a new agent will be fabricated accordingly.

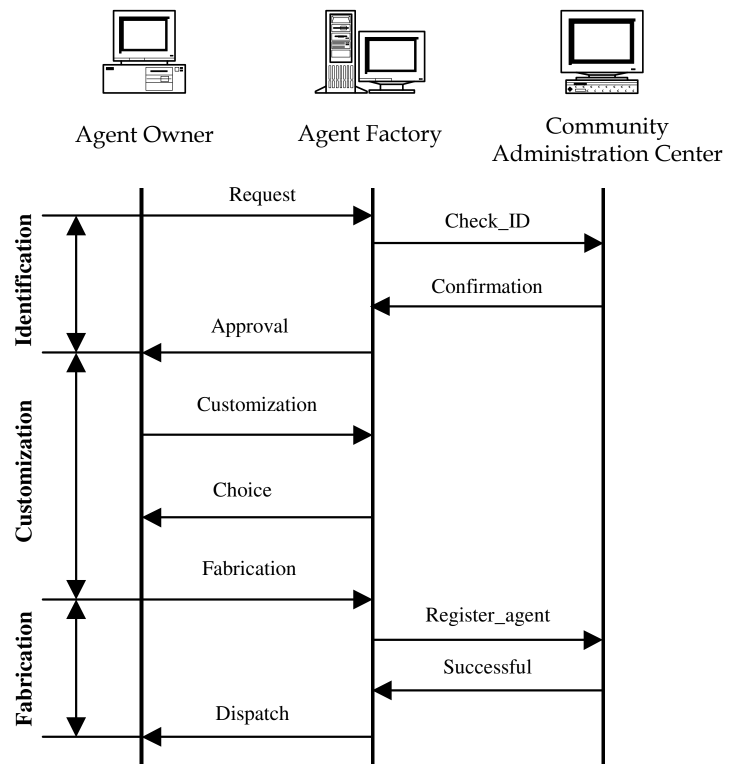


Figure 5. Formalities of Agent Fabrication

After an agent is created successfully, the agent factory stores its information for future use. The factory is also responsible for registering the new agent with the CAC, which is done by sending a *Register_agent* message to the CAC. The CAC will generate a certificate for the agent, record it in its roster, and return a *Successful* message to the agent factory. The agent factory will integrate the certificate into the new agent's identity module and reply to the agent owner with a *Dispatch* message. The newly born agent is then dispatched. At this stage, the fabrication procedure has completed successfully and the agent owner can employ the agent for intended tasks.

4. IMPLEMENTATION

The implementation of agent fabrication includes the development of many entities such as community administration center, agent factory, agent owner, product server, as well as the agent fabrication procedures.

Java is chosen as the implementation language as it bundles a lot of attractive features including portability, stability, and security. Java's portable byte code feature enables an agent to roam anywhere across the network and resume its operation at any host that supports Java Virtual Machine, regardless of the underlying operating system. In addition, with Java's multithreading facility, an agent can be allowed to execute on a separate thread, independently of other agents executing on the same host.

4.1 Implementation of the SAFER Entities

We have implemented a prototype of SAFER framework in which agent fabrication is an essential part. We have implemented a community that includes one agent factory, one community administration center, product servers, and several agent owners. Brokering agents have been fabricated successfully in the agent factory according to the formalities described earlier.

Figure 6 shows the class inheritance diagram of the main entities involved in agent fabrication. All entities in SAFER are extended directly or indirectly from the *Entity* class, which defines the basic features for all entities such as identity, description, certificate, and corresponding manipulation methods. All extended entities automatically inherit the basic feature of the base class, plus some additional features individually. For an ECAgent class, it also implemented Serializable interface. Thus, an ECAgent object can be converted to a byte stream and sent across the network.

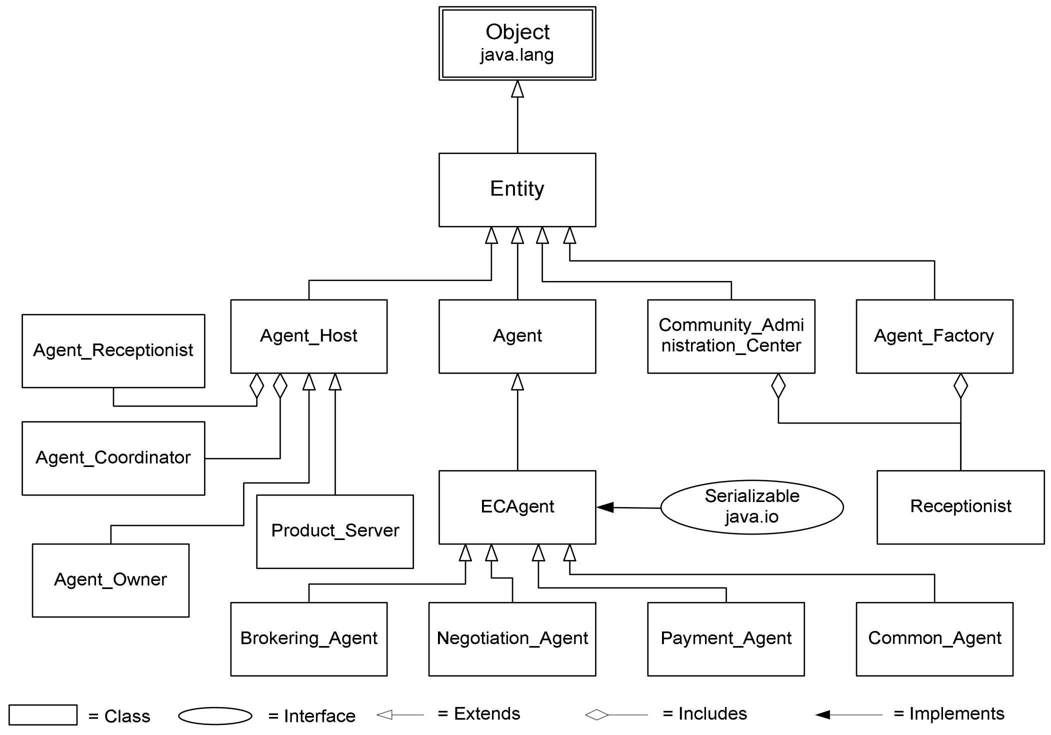
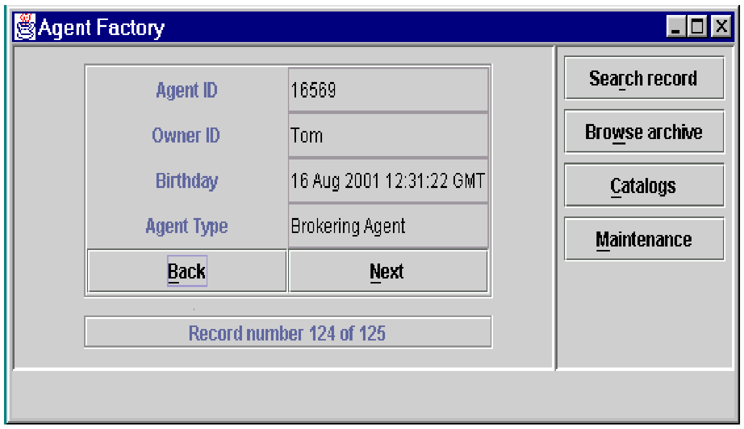


Figure 6. Class Inheritance Diagram

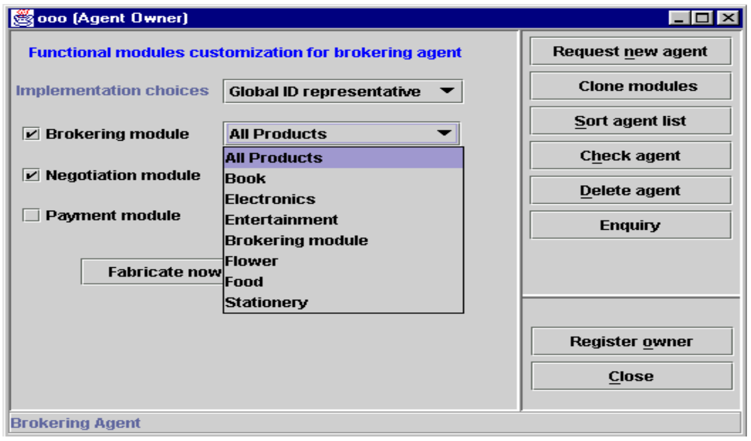
Figure 7 shows the interface for an agent factory. The interface looks somewhat simple, as most functions of an agent factory are automatic and invisible to users. Apart from these automatic functions, Figure 7 shows the interface for other functions, such as searching agent records, browsing factory archive, viewing agent catalogs, and maintaining the agent factory. The factory archive stores records of all the agents that were fabricated in the factory. Therefore, users can search agent records using keywords such as agent ID, owner ID, and agent type.



Agent ID: 16569
Owner ID: Tom
Birthday: 16 Aug 2001 12:31:22 GMT
Agent Type: Brokering Agent
Record number 124 of 125

Figure 7. Screenshot of a User Browsing Agent Records in the Agent Factory

Among the three stages of agent fabrication, the customization stage is the most important and complicated. Figure 8 depicts the scenario that an agent owner is customizing a new brokering agent. Among a variety of module choices, the owner can pick up modules according to his preferences. Some modules are indispensable, while some are optional. A user can also specify parameters in some modules after he chooses these modules. After fixing the customization information, the agent owner can request the agent factory to continue the fabrication process.



Functional modules customization for brokering agent
Implementation choices: Global ID representative
[✓] Brokering module: All Products
[✓] Negotiation module
[ ] Payment module
Brokering Agent

Figure 8. Screenshot of Module Customization during Fabrication

4.2 Implementation of the Task Queue and Scheduler

For ensuring the sequential processing of tasks inside, the task queue maintains five attributes, namely *front*, *rear*, *noOfTask, maxSize*, and *queue*. The first four are all integers. The *front* attribute locates the earliest task inside the queue, while the *rear* attribute points to the latest task. The *noOfTasks* attribute counts the number of outstanding tasks inside the queue. The *maxSize* attribute is set as the maximum number of tasks that a task queue can hold. The *queue* attribute is an array of scheduled tasks.

The task queue also contains some methods to manage the task queue. The *get()* method takes out the task at the front location if the queue is not empty, and move the *front* pointer to the next task. The *put(Task)* method is responsible for adding the new task at the rear location if the queue is not full, and moves the *rear* pointer to point to the new task. The *peek()* method views the task at the *front* location without taking it out. The *isFull()* and *isEmpty()* methods are to check whether the task queue is full or empty respectively.

When an agent gets an assignment from its owner, the *compose()* method of the scheduler will be called to plan the agent's tasks to accomplish the assignment. Whenever an agent gets its own thread of control, it will invoke the *performTask()* method of the scheduler. The *performTask()* method is responsible for performing the following tasks:

- Checking whether the task queue is empty or not by using the *isEmpty()* method of the task queue.

- Taking out the current task with the *getTask()* method and invoking it. If the task queue is empty, the scheduler will stop its executing tread.
- Checking for dependency with *checkDependency()* method if the current task is unsuccessful.

The *checkDependency()* method checks the dependency between the unsuccessful task and its dependent tasks by searching through the task queue. The scheduler is responsible for removing the dependent tasks of a failed task. When a task fails, the scheduler will check whether the failed task has its dependent tasks or not. If so, the scheduler will remove the dependent tasks from the task queue. The *addTask()* method will use *put()* to add a new task into the task queue.

4.3 Implementation of a Product Server

We have implemented a product server that serves brokering agents with product information. The architecture of a product server is shown in Figure 9. The main components are agent coordinator, agent receptionist, database handler, and various product databases.

The agent receptionist runs at the product server, assuming responsibilities to protect the server against malicious agents. The agent receptionist will verify the identity of an incoming agent at the CAC before allowing it to run at the product server. Furthermore, it will also be responsible for assigning a communication channel for the incoming agent with one of its unused TCP ports. A new agent runtime environment will also be created for the incoming agent to execute. With the new agent runtime environment, the incoming agent will get its own thread for execution and begin to perform its assignment.

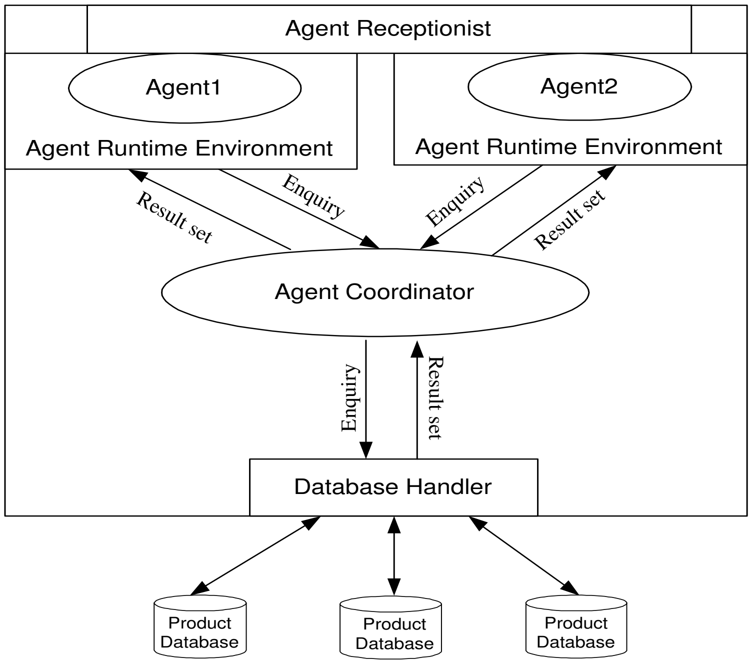
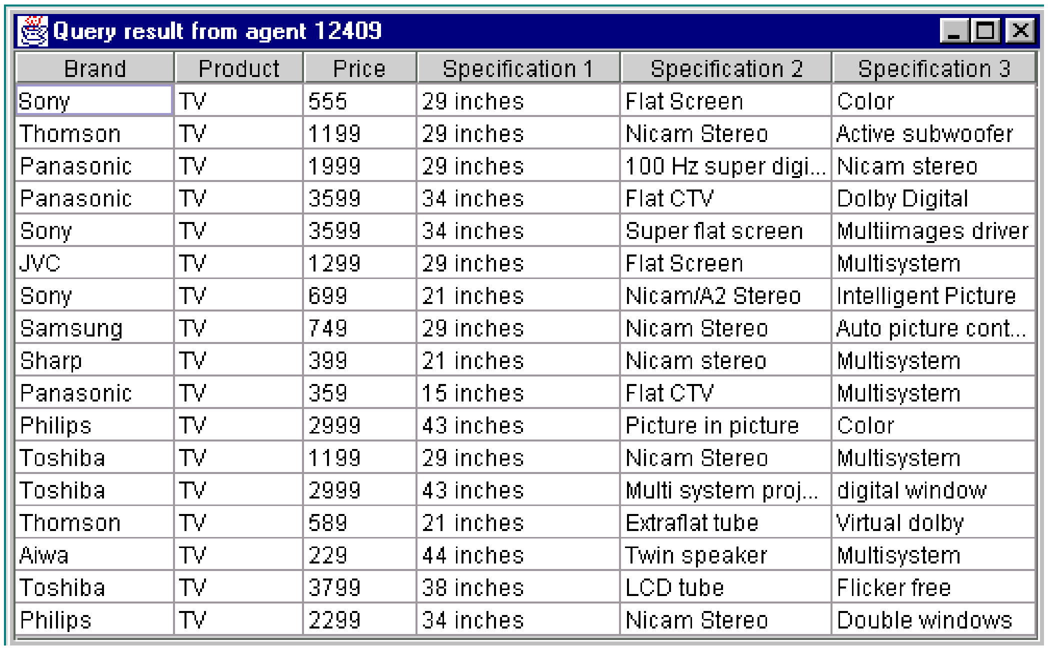


Figure 9. Product Server Architecture

The agent requests for services through the agent coordinator since they are not allowed to access the local product databases directly. The agent coordinator will locate the databases and forward the agent's requests to the database handlers. Therefore, the internal components in a product server are protected from direct access, which will be more secure for a product server.

In order to handle multiple requests concurrently, the agent coordinator will create a new message handler for each incoming request. In this way, the agent coordinator can pass the socket connection to a message handler and waits for other incoming requests, while the message handler will be responsible for fulfilling the request.



Brand	Product	Price	Specification 1	Specification 2	Specification 3
Sony	TV	555	29 inches	Flat Screen	Color
Thomson	TV	1199	29 inches	Nicam Stereo	Active subwoofer
Panasonic	TV	1999	29 inches	100 Hz super digi...	Nicam stereo
Panasonic	TV	3599	34 inches	Flat CTV	Dolby Digital
Sony	TV	3599	34 inches	Super flat screen	Multiimages driver
JVC	TV	1299	29 inches	Flat Screen	Multisystem
Sony	TV	699	21 inches	Nicam/A2 Stereo	Intelligent Picture
Samsung	TV	749	29 inches	Nicam Stereo	Auto picture cont...
Sharp	TV	399	21 inches	Nicam stereo	Multisystem
Panasonic	TV	359	15 inches	Flat CTV	Multisystem
Philips	TV	2999	43 inches	Picture in picture	Color
Toshiba	TV	1199	29 inches	Nicam Stereo	Multisystem
Toshiba	TV	2999	43 inches	Multi system proj...	digital window
Thomson	TV	589	21 inches	Extraflat tube	Virtual dolby
Aiwa	TV	229	44 inches	Twin speaker	Multisystem
Toshiba	TV	3799	38 inches	LCD tube	Flicker free
Philips	TV	2299	34 inches	Nicam Stereo	Double windows

Figure 10. Result Set Presented by a Brokering Agent

We have created a number of Microsoft Access databases in a product server for agents to search, by submitting their queries using SQL (Structured Query Language). Figure 10 shows that a brokering agent has retrieved product information from a product server, and present the result set to its owner when it returns to the owner host.

4.4 A Prototype of Virtual Marketplace

We have also implemented a prototype of a virtual marketplace where buyer agents and airline agents can negotiate the price of air-tickets. The detailed design can be found in

[23]. The integration of the virtual marketplace into the SAFER framework is in active progress.

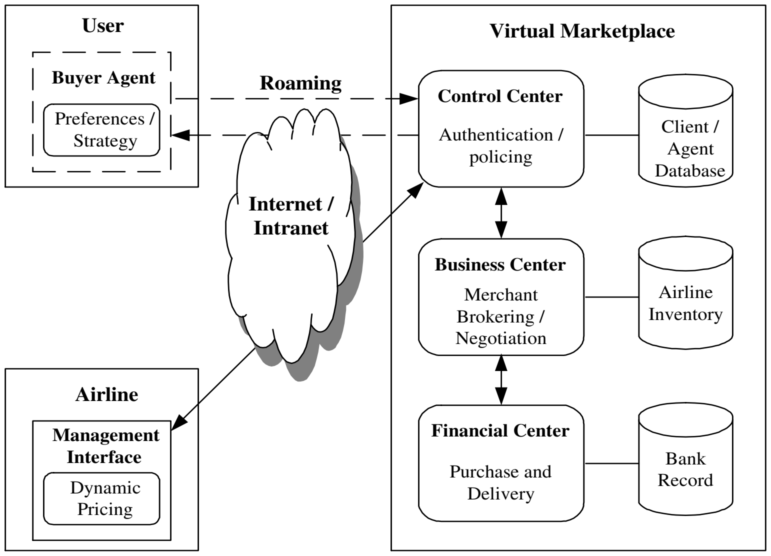


Figure 11. A Prototype of Virtual Marketplace

Figure 11 shows the architecture of a virtual marketplace. It consists of three separate elements, namely, control center, business center, and financial center. Seller agents are permanent entities residing in the marketplace and they belong to individual airlines, and airline companies can manage their agents via management interfaces. Buyer agents act on behalf of users who are interested in purchasing air tickets which best match their preferences. They will meet seller agents in the marketplace, negotiate with them, and even make transactions if applicable. A user can set his/her preferences based on the details like flight time, preferred airlines, etc. Certain parameter such as departure time also has a flexibility rating. The user has the option of choosing among different flexibility settings that are used to determine an acceptable range for that particular parameter. After setting the desired preferences, the user is required to customize the buyer agent's negotiation strategy. This includes setting an initial offer price, the maximum allowable price, and a choice of three time-based price-adjustment functions. The user can then proceed to dispatch his buyer agent into the marketplace. After being authenticated by the control center, the buy agent will be matched with several seller agents according to the preference settings. A negotiation session will be initiated through the help from a proxy agent designated by the marketplace. Both the buyer agent and seller agents have their own negotiation strategies to propose offers or counter-offers. The negotiation session will last until both sides agree on the price or either side quits. If they finally reach a deal, they will conduct the transaction in the financial center and the user will be informed of the transaction details.

5. DISCUSSIONS

When designing the scheduler and task queue, we assume that the itinerary of a roaming agent is planned in advance, so that the task queue can be determined *a priori*. The scheduler is only responsible for the scheduling of pre-defined tasks and it cannot change or reschedule an agent's itinerary. However, it may be necessary to create new tasks or reschedule the task queue due to some conditions causing an itinerary change. This also means we need to embed agents with such dynamic task creation and rescheduling capability so that it can cope with new events and readjust its itinerary when required.

We present an agent life cycle model from the viewpoint of an agent (or agent owner) itself. Actually, we could design the lifecycle model from a host's point of view, with different focus. For example, a host may be more interested in whether an agent is 'running'; or has 'roamed', rather than whether a foreign agent is 'dead'. When an agent has ‘roamed’ to another host, the current host could then archive the agent’s data to disk if it still expect the agent to return in the near future. Otherwise, the host could choose to delete the agent’s records when the agent has roamed away forever.

With the feature of GID, agent body will be lighter and network bandwidth requirements will be reduced during agent roaming. But there is a basic requirement that the hosts visited by such agents should be able to download required class packages from the agent factories. Such downloading may not be possible if the hosts are barred from factory access due to firewall constraints.

With the factory-based approach for agent fabrication, agent software upgrade is made much easier. For example, an agent factory may update modules in an agent and consequently assign a new version number, without disposing of a GID-based agent. It is also easier to ensure the integrity of agents, as they are fabricated by authorized agent factories. For instance, an agent digest may accompany a roaming agent, and a certain cryptographic method such as PKI (Public Key Infrastructure) can be used for integrity protection and certificate authentication.

6. CONCLUSIONS

This paper presents a factory-based agent fabrication scheme which aims to provide a convenient and safe approach to create agents for various e-commerce applications. SAFER is introduced as a basic framework for agent fabrication. A generic modular structure has been proposed to facilitate agent fabrication. A rich set of off-shelf modules provided by the agent factory enables users to customize agents for delegated tasks.

Agent life cycle is presented to control the state transitions of an agent. Agent fabrication formalities are carefully designed to facilitate the fabrication process and strengthen agent security and trustworthiness.

Currently, the entities involved in agent fabrication, including the community administration center, agent factory, agent owner, and product server, have been implemented with Java successfully. Brokering agent is employed as a practical agent type to demonstrate our design and implementation for agent fabrication. A product server is implemented to evaluate the functionality of brokering agents. A prototype of virtual marketplace is also implemented as a potential application for agent-based e-commerce.

We plan future enhancements as follows. Firstly, some agent communication language such as FIPA-ACL can be used to standardize agent communication [9]. Secondly, a directory service, which maintains information about all the services offered, may be implemented to help agents locate the service providers. Thirdly, coordination and cooperating mechanisms will be implemented to enable the collaboration among multi-agents to achieve common goals.

REFERENCES

[1] Bradshaw, J.M. *Software Agent,* MA: MIT Press, 1997.

[2] Chavez, A. and Maes, P. Kasbah: an agent marketplace for buying and selling goods. In *Proceedings of First International Conference on Practical Application of Intelligent Agents and Multi-Agent Technology*, London, 1998, 75-90.

[3] Collis, J., Ndumu, D., Nwana, H., and Lee, L. The Zeus agent building toolkit, *BT Technology Journal,* 16(3), 1998.

[4] Corradi, A., Montanari, R., and Stefanelli, C. Mobile agents integrity in e-commerce applications. In *Proceedings of 19th IEEE International Conference on Distributed Computing Systems*, 1999, 59-64.

[5] Cusumano, M.F. The software factory: a historical interpretation, *IEEE Software*, March, 1989, 23-30.

[6] Dignum F., and Cortés, U. (eds). Agent-mediated electronic commerce III: current issues in agent-based electronic commerce systems. *Lecture notes in artificial intelligence*. Berlin: Springer, 2001.

[7] Doorenbos, R., Etzioni, O., and Weld, D. A scalable comparison-shopping agent for the World Wide Web. In *Proceedings of the First International Conference on Autonomous Agents,* Marina del Rey, CA, 1997, 39-48.

[8] Fernstrom, C., Narfelt, K.H., and Ohlsson, L. Software factory principles, architecture, and experiments, *IEEE Software*, 9(2), 1992, 36 –44.

[9] FIPA, Foundation of Intelligent Physical Agents, http://www.fipa.org.

[10] Glushko, R.J., Tenenbaum, J.M., and Meltzer, B. An XML framework for agent-based e-commerce. *Communications of the ACM*, 42(3), 1999, 106-114.

[11] Greenberg, M.S., Byington, J.C., and Harper, D.G. Mobile agents and security. *IEEE Communications Magazine*, 36(7), 1998, 76-85.

[12] Guan, S.U. and Yang, Y. SAFE: secure-roaming agent for e-commerce. In *Proceedings the 26th International Conference on Computers & Industrial Engineering*, Melbourne, Australia, 1999, 33-37.

[13] Guan, S.U., Zhu, F.M., and Ko, C.C. Agent fabrication and authorization in agent-based electronic commerce. In *Proceedings of International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce*, Wollongong, Australia, 2000, 528-534.

[14] Guttman, R.H. and Maes, P. Agent-mediated negotiation for retail electronic commerce. In *Agent Mediated Electronic Commerce: First International Workshop on Agent Mediated Electronic Trading*, Springer, Berlin, 1999, 70-90.

[15] Hendler, J. Agents and the Semantic Web. *IEEE Intelligent Systems*, 16(2), 2001, 30-37.

[16] Hua, F. and Guan, S.U. Agent and payment systems in e-commerce, *Internet Commerce and Software Agents: Cases, Technologies and Opportunities,* Rahman, S.M. and Bignall, R.J. (ed.), Idea Group, PA, 2000, 317-330.

[17] Krishna, V. and Ramesh, V.C. Intelligent agents for negotiation in market games, part2: application. *IEEE Transactions on Power Systems*, 13(3), 1998, 1109-1114.

[18] Lange, D.B. and Oshima, M. *Programming and Deploying Mobile Agents with Java Aglets*, Mass.: Addison-Wesley, 1998.

[19] Lassila, O., Van Harmelen, F., Horrocks, I., Hendler, J., and McGuinness, D.L. The semantic Web and its languages. *IEEE Intelligent Systems*, 15(6), 2000, 67 –73.

[20] Lee, J.G., Kang, J.Y., and Lee, E.S. ICOMA: an open infrastructure for agent-based intelligent electronic commerce on the Internet. In *Proceedings of International Conference on Parallel and Distributed Systems*, 1997, 648-655.

[21] Maes, P. Agent that reduce work and information overload. *Communication of the ACM*, 37(7), 1994, 31-40.

[22] Marques, P.J., Silva, L.M., and Silva, J.G. Security mechanisms for using mobile agents in electronic commerce. In *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*, 1999, 378-383.

[23] Ng, C.H., Guan, S.U., and Zhu. F.M., Virtual Marketplace for Agent-based Electronic Commerce, to appear in *Architectural Issues of Web-Enabled Electronic Business,* PA: Idea Group, 2002.

[24] Reticular Systems, Inc. AgentBuilder: an integrated toolkit for constructing intelligent software agents, revision 1.3, *http://www.agentbuilder.com/.,*1999.

[25] Wang, T.H., Guan, S.U., and Chan, T.K. Integrity protection for code-on-demand mobile agents in e-commerce. To appear in *Journal of Systems and Software,* 2001.

[26] Wang, T.H., Guan, S.U., and Ong, S.H. An agent based auction service for electronic commerce, in *Proceedings of International ICSC Symposium on Interactive and Collaborative Computing (ICC'2000)*, Australia, 2000.

[27] Wurman, P.R., Wellman, M.P., and Walsh, W.E. The Michigan Internet AuctionBot: a configurable auction server for human and software agents. In *Proceedings of the Second International Conference on Autonomous Agents,* Minneapolis, USA, 1998, 301-308.

[28] Zhu, F.M. and Guan, S.U. Evolving software agents in e-commerce with GP operators and knowledge exchange, in *Proceedings of the 2001 IEEE Systems, Man, and Cybernetics Conference*, Tucson, USA, 2001.

[29] Zhu, F.M., Guan, S.U., and Yang, Y. SAFER E-Commerce: Secure Agent Fabrication, Evolution & Roaming for E-Commerce. *Internet Commerce and Software Agents: Cases, Technologies and Opportunities,* Rahman, S.M. and Bignall, R.J. (ed.), PA: Idea Group, 2000, 190-206.