Agent Fabrication and Its Implementation for Agent-based Electronic Commerce

Sheng-Uei Guan and Fangming Zhu
Department of Electrical and Computer Engineering
National University of Singapore
10 Kent Ridge Crescent, Singapore 119260
{eleguans@nus.edu.sg}

Abstract In the last decade, agent-based e-commerce has emerged as a potential role for the next generation of e-commerce. How to create agents for e-commerce applications has become a serious consideration in this field. This paper proposes a new scheme named agent fabrication and elaborates its implementation in multi-agent systems based on the SAFER (Secure Agent Fabrication, Evolution & Roaming) architecture. First, a conceptual structure is proposed for software agents carrying out e-commerce activities. Furthermore, agent module suitcase is defined to facilitate agent fabrication. With these definitions and facilities in the SAFER architecture, the formalities of agent fabrication are elaborated. In order to enhance the security of agent-based e-commerce, an infrastructure of agent authorization and authentication is integrated in agent fabrication. Our implementation and prototype applications show that the proposed agent fabrication scheme brings forth a potential solution for creating agents in agent-based e-commerce applications.

Keywords software agent, e-commerce, agent fabrication

1. Introduction

The Internet and electronic commerce (e-commerce) are revolutionizing our concept and behavior in doing business. Nowadays, more and more companies and customers are getting used to selling and purchasing online. As a result, the revenue of e-commerce in the world is increasing tremendously.

E-commerce can offer a lot of advantages such as lower operating cost, higher accessibility, and broader services. However, there are some barriers blocking the road to success, which include overload of information, difficulty in searching, lack of negotiation infrastructure, etc. Therefore, e-commerce demands advanced technologies as support. Software agent seems to be the excellent candidate with its properties of intelligence, autonomy, and mobility. Agent-based e-commerce has emerged and become the focus of the next generation of e-commerce. In this new approach, software agents act on behalf of customers to carry out delegated tasks automatically. They have demonstrated tremendous potential in conducting various tasks in e-commerce, such as comparison shopping, negotiation, payment, etc.

Most research work in literature focuses on issues such as traceability, integrity, and security of agent systems [Corradi (1999), Greenberg (1998), Marques (1999)], while very little work touches the field of agent creation. However, creation of agents should be essential, since it is the starting point and the way of agents being created will be directly related to concerns about agent security.

MIT Media Lab's Kasbah [Chavez (1998)] is an online marketplace for buying and selling goods. A user can create a buyer agent, provide it with a set of criteria, and dispatch it into marketplaces. The Minnesota AGent Marketplace Architecture (MAGMA) [Tsvetovatyy (1997)] is a prototype for a virtual marketplace targeted toward items that can be transferred over the Internet. Agents can register with a server that maintains unique identifiers for agents. The Michigan Internet AuctionBot [Wurman (1998)] is an auction server which

facilitates the connection of potential buyers and sellers. Agents can place bids, create auctions, request auction information, or review their accounts. Since these projects are designed with their specific goals such as negotiation or information collection, agents inside them are function-oriented and only supported by the specific marketplaces. Moreover, users are only guided to fill in their particulars, and have little influence on the creation of new agents. This means that users cannot embody individual preferences in their agents, and they are given little chance to customize agents.

In recent years, there has been a new research stream which aims to help users creating agents by providing certain frameworks and development toolkits. Some of them even have been provided as commercial packages. AgentBuilder [Reticular (1999)] is such an integrated tool suite for constructing intelligent software agents. It is designed to provide agent software developers with an integrated environment to quickly and easily construct intelligent agents. Concordia [Wong (1997)] is another full-featured framework for the development and management of agent applications. The work of creating new agents in Concordia is left to users, while Concordia provides users with an agent development guide which covers the issues of programming agents in great details. Zeus [Collis (1998)] provides a toolkit for creating generic collaborative agents. It comprises of three functional groups, i.e. an agent component library, an agent-building tool, and an agent visualization tool. It enables users to visually specify their significant attributes in creating agents. Users can also view, analyze, or debug agents with the visualization tool. Aglet [Lange (1998)] provides Aglet Software Development Kit (ASDK) for creating and managing aglet, mobile Java agent. ASDK includes API (Application Programming Interface) packages with which users can implement a platform independent aglet. Users can also override necessary subclasses of Aglet to meet aglet behavior for different assignments.

However, these 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. For example, if a user needs to employ an agent for some negotiation tasks, at least he needs to program part of the agent such as negotiation strategies. But most users do not have such programming skills, and a simple usage of toolkits may even be difficult for them. Furthermore, e-commerce agents created with different toolkits can only be supported by compatible marketplaces, and this situation can also lead to lack of interoperability, which may result in disorder and cause difficulty in communication among agents. These limitations is due to the fact that the original intention of these toolkits is to assist users in developing generic intelligent agents, not meant to meet the particular demands of e-commerce applications.

The agents in e-commerce applications have their distinctive characteristics, compared to agents for other purposes. For example, e-commerce has much wider user bases. So it should be easy for users (especially novice users) to create and employ agents. Furthermore, e-commerce cannot be widely accepted unless the related security concerns can be relieved, for agent-based e-commerce this means agents should be secure and trustworthy. Our agent factory approach has reduced the security hazards that could be possible if agents can be fabricated by potential hackers. Therefore, these special demands of e-commerce agents have motivated us to set up an architecture/mechanism to facilitate agent fabrication.

As agents roam to visit various hosts and operate in those hosts, some security concerns would arise naturally. How can hosts identify incoming agents? How can agents ensure that their confidential information is not being revealed during operations in hosts? On the one hand, hosts cannot trust incoming agents belonging to unknown owners, because malicious agents may launch attacks on the hosts and other visiting agents. On the other hand, agents may also have concerns on the reliability of hosts and will be reluctant to expose their secrets to distrustful hosts. To build bilateral trust in an e-commerce environment, an infrastructure of agent authorization and authentication should be well designed and implanted into agents when they are created.

In order to alleviate the above concerns, a new agent fabrication scheme, which is factory-based, has been proposed and integrated into the SAFER architecture (Secure Agent Fabrication, Evolution & Roaming) [Zhu (2000)]. The objective of our scheme is to provide a

convenient and safe approach to create agents for various 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. The agents created would have a uniform structure that can facilitate communication and collaboration among agents. Further, users can be alleviated from the laborious work of programming, and agents are fabricated in a more systematic and standardized way. Hosts can be relieved from the risks of accommodating agents fabricated by hackers. In addition, the scheme also aims to enhance the security of agent-based e-commerce by integrating an infrastructure of agent authorization and authentication into agent fabrication. Our scheme of agent fabrication sheds light on a new way of creating agents in agent-based e-commerce applications.

This paper is organized as follows. Section 2 briefly introduces the SAFER architecture and its main components involved in agent fabrication. Section 3 elaborates agent conceptual structure, agent module suitcase, and fabrication formalities. An infrastructure of agent authorization and authentication is illustrated in section 4. Implementation work is presented in section 5. Section 6 presents two prototype applications. Section 7 concludes the paper.

2. Architecture of SAFER

SAFER is an infrastructure designed to serve agents in e-commerce and establish necessary mechanisms to manipulate them. The main objective of SAFER is to construct an open, dynamic and evolutionary agent system for agent-based e-commerce, incorporating agent fabrication [Guan (2000)], evolution [Zhu (2001)], and roaming [Guan (1999)]. Agent fabrication is one of the fundamental parts in the SAFER Architecture.

Agent communities are basic units in SAFER. Each SAFER community can possess a set of facilities and entities as described in Figure 1. However, Figure 1 only lists typical entities in a community, as some communities may have more or less entities than those depicted. For instance, the number of agent owners can be varying all the time, and several communities may share one factory.

Since this paper focuses on agent fabrication, only the involved components are briefly introduced here. These components include agent factory, community administration center, agent owner, and agent itself. The detailed description is provided in [Zhu (2000)].

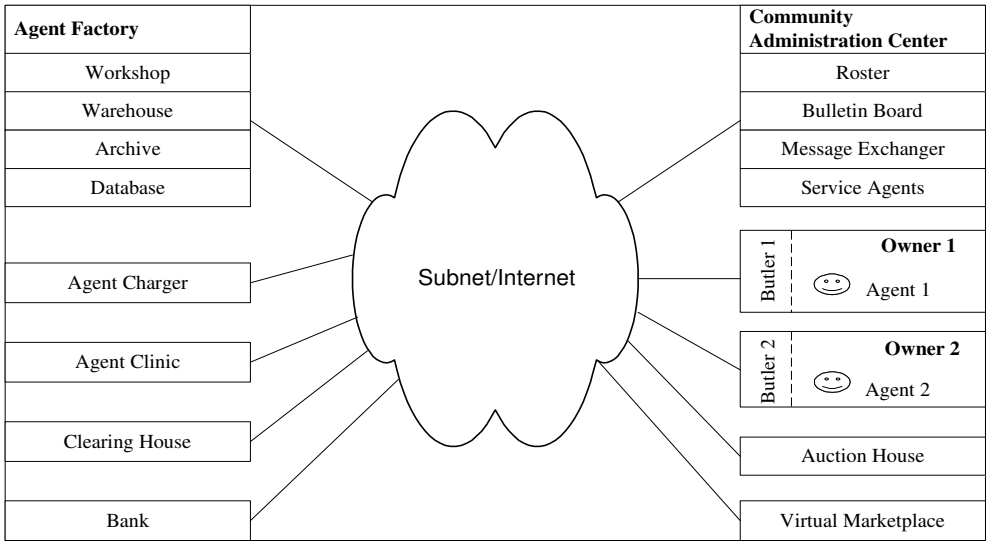

Figure 1. SAFER agent community

Agent factory is the kernel of SAFER, especially considering its role in agent fabrication. It undertakes the primary task of creating agents. An agent factory assembles new agents with desirable functionality according to the customizations from owners. In addition, an agent factory has the responsibility to fix and check agents, which is an indispensable function of agent integrity and security.

Community Administration Center (CAC) is responsible for administrative matters in the community, coordinating and facilitating activities of the entities in the community. It aims to ensure smooth routine operations and security of the whole community. CAC maintains a roster for the community, which includes basic information of registered owners and agents. This roster is updated periodically.

Agent owner stands at the top of the SAFER hierarchy, since he has the priority and responsibility for all his agents. He controls his agents from creation to termination. Each owner should register successfully in CAC before he can have access to the facilities in the community. To relieve his burden, an owner can authorize his butler to handle most of his tasks.

Agents play a central role in SAFER, as facilities in SAFER serve agents in one way or another. Agents are fabricated by authorized agent factories and customized by their owners. To identify itself, each agent has a unique ID issued by the agent factory. An agent can roam from one host to another to carry out various tasks, crossing the boundary of communities.

3. Agent Fabrication

3.1 Conceptual structure for e-commerce agent

In order to facilitate agent fabrication, a conceptual structure for e-commerce agent is proposed, which is shown in Figure 2. There are four functional layers in the conceptual structure, namely, communication layer, identity layer, security layer, and application layer [Guan (2000)].

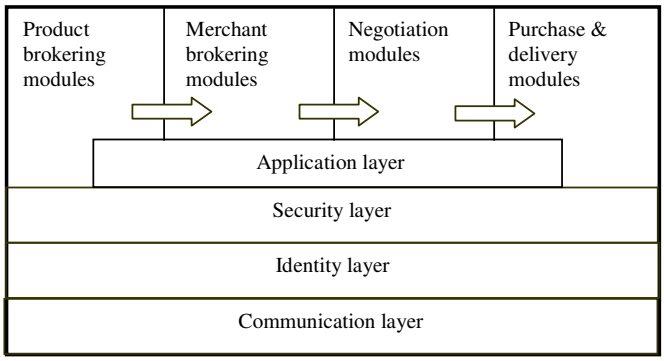

Figure 2. Conceptual structure for e-commerce agent

The communication layer comprises some message modules that are used for communicating with external entities such as owners, agent factories, as well as other agents. The identity layer includes personal information of the agent. The security layer consists of modules concerning mechanisms of security and self-protection, such as implementation of cryptography tools and measures to deal with attacks from malicious agents or hosts.

The application layer is filled with various functional modules. According to the nomenclature of Maes' group in the MIT Media Lab [Guttman (1999), Maes (1994)], the common commerce behavior can be described with Consumer Buying Behaviour (CBB) model, which consists of six stages, namely, need identification, product brokering, merchant brokering, negotiation, purchase & delivery, and product service & evaluation. Since the first

and last stages are atypical and dispensable in a practical shopping situation, only four stages are included in our conceptual structure for e-commerce agents, as shown in Figure 2.

Actually, one agent can possess one to four modules in the application layer. Thus, a typical e-commerce task may be solely completed by one agent or through collaboration of several agents. The arrows between modules in Figure 2 stand for information flowing from the previous module to the next module. For instance, after retrieving information from chosen merchants, the merchant brokering module will pass addresses of these merchants to the negotiation module for continuing the tasks. Therefore, if all the stages are carried out by one single agent, these information flows will happen internally. While in the scenario of collaboration, information flow will take place among agents with the help of communication modules.

3.2 Agent module suitcase

Modularization is a theme that runs through the whole process of agent fabrication. Various modules for different types of agents are stored in agent factories. These modules are combined to form an agent during fabrication. But not all the modules in a factory will be assembled into an agent. An agent is just like a suitcase, into which necessary modules can be loaded according to the requests from its owner and guidelines for fabrication. As a matter of fact, the size of an agent is essential for its efficiency and fitness. An agent with redundant modules will be less efficient, because a lot of time is wasted on transferring its heavier body during roaming. On the other hand, if an agent lacks the necessary modules, it cannot fulfill even the basic functions. Therefore, how to achieve an optimal module combination is the pursuit of agent factories and owners.

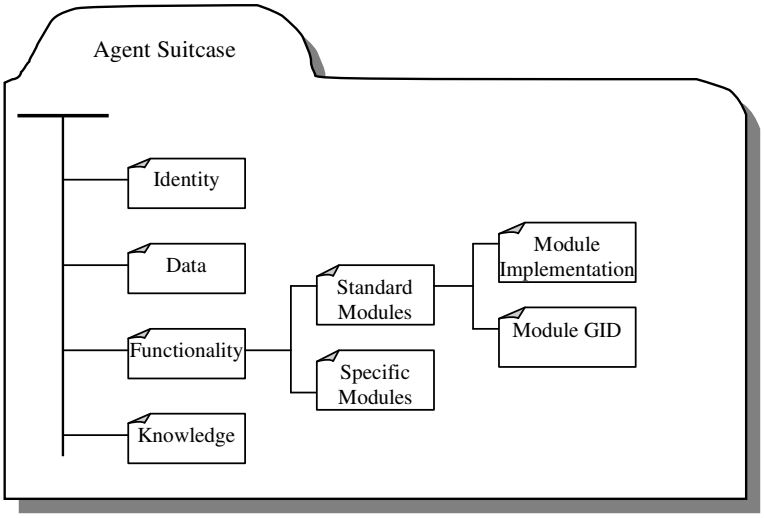

Figure 3. Agent module suitcase

A general agent module suitcase is defined as in Figure 3. A suitcase is composed of four kinds of modules, i.e., identity, data, knowledge, and functionality. The identity module contains basic elements of the identity of an agent, such as agent ID, certificate, timestamp, agent-digest, etc. The data modules are to store information collected from hosts, parameters used in functionality modules, as well as logs of the agent activities. The knowledge modules store the knowledge base to support analysis and decision-making. The most important part of an agent is the functionality modules. They comprise specific and standard modules. Each specific module, e.g. negotiation module, is customized by its owner for different purposes. Specific modules constitute the variable components of an agent. For the standard modules,

SAFER provides two choices, i.e. direct module implementation and virtual module implementation with Global ID (GID). GID is a string representing a standard module in agent factories. GIDs can be placed into an agent suitcase instead of the real implementations to decrease the volume of an agent. In SAFER, each merchant host keeps a database of standard module implementations. Whenever an agent visiting a host needs to make use of standard functions which are represented by GIDs, the host will simply load the module implementations from its database if available, according to the GID in the agent suitcase. Even if the implementation associated with certain GID cannot be found in the database, the host can download it from agent factories. The tradeoff on binding to either direct or virtual implementation is determined by each owner. An example of agent suitcase is illustrated in Figure 4.

```
Agent_Tom {
        Identity {String    AgentID;
                  String    Timestamp;
                  String    Certificate;
                  String    Agent_Digest;
                  };
        Functionality       {
                  Void      Com_Message1();
                  Void      Security_Check();
                  Void      Negotiation();
                            };
        Knowledge           {
                  Void      Product_Ontology();
                            };
        Data      {
                  Void      DataFromHost();
                  Void      Nego_Strategy_Para();
                  Void      Log();
                  };
        }
```

Figure 4. An example of agent suitcase

3.3 Agent fabrication formalities

In SAFER, agent fabrication obeys a set of strictly prescribed formalities. These formalities involve some SAFER facilities, including community administration center (CAC), agent factory, and agent owner, which have been introduced in section 2. The process of agent fabrication comprises three stages, namely, identification, customization, and fabrication, as illustrated in Figure 5.

The fabrication process is initiated by an agent owner. First, the agent owner sends a *request* message to an agent factory. The *request* message contains information about the identities of the owner and community. When the agent factory receives the request, it contacts the corresponding CAC with *check-ID* message to check the identity of the agent owner. Then CAC looks up the identity of the owner in the roster, which has basic information of registered owners and is updated periodically, as mentioned in section 2. After CAC ensures that the owner has registered before, it returns *confirmation* to the agent factory. Then the agent factory informs the owner of the *approval* of his fabrication request. The agent owner then requests to start the stage of *customization*. During customization, the agent factory provides many *choices* such as agent type, module combinations, etc. Having all the requirements settled, the agent factory then starts the assembly procedures following the *fabrication* instruction from the agent owner. After an agent is successfully fabricated, the agent factory will update its own database and send *register-agent* to CAC who will add the

information of this new agent into the roster. In addition, the fabrication event will be recorded in the archive of agent factory. As soon as the agent factory gets back the *successful* message from CAC which indicates a successful update, the infant agent will be *dispatched* to the owner who can deploy the new agent to undertake intended tasks.

In order to fabricate a new agent successfully, the fabrication process must pass through all three stages successfully. Sometimes accidents may occur unexpectedly. For instance, the agent owner and the agent factory may not have reached an agreement in customization phase, or the agent owner has not registered yet, or messages are lost during transfer. These will result in termination of the fabrication procedure midway.

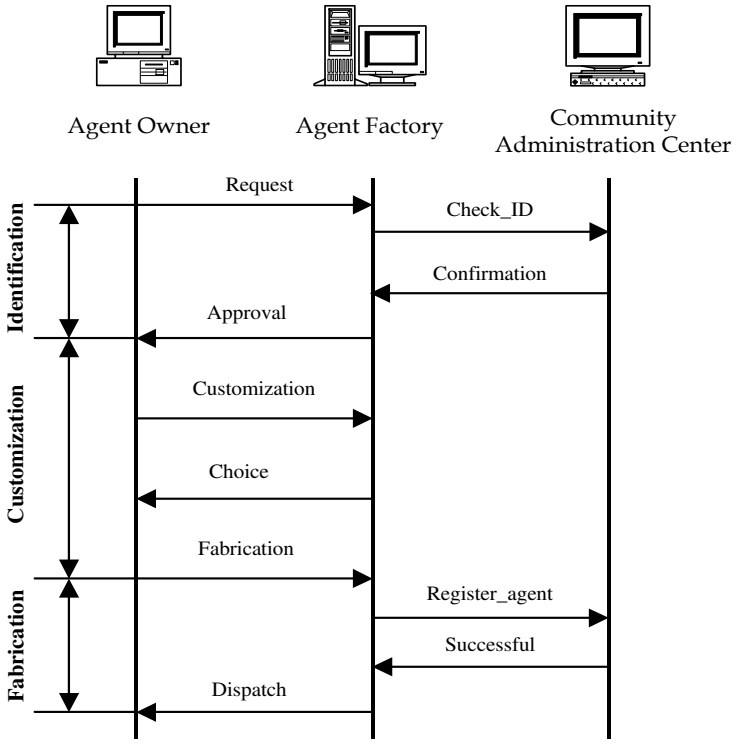

Figure 5. Agent fabrication formalities

4. Agent Authorization and Authentication

In order to enhance security, an infrastructure of agent authorization and authentication is designed and integrated into agent fabrication. Agent authorization builds the identity of an agent, while authentication checks the identity.

Agent authorization in SAFER is controlled with a tower structure. Agent community is at the top of the tower, and agent factory, agent owner and software agent lie in lower positions in sequence. The authorization procedure is from top to bottom. Entities in higher positions have the information of entities in lower positions and take responsibility of authorizing the lower ones. In SAFER, one of the most important procedures during fabrication is to assign a unique ID to each newborn agent. The agent ID is created by concatenating four strings, namely, Community ID, Factory ID, Owner ID, and Fabrication Series Number. Since these four components are all unique, the resulting agent ID is ensured to be unique too.

PKI (Public Key Infrastructure) is introduced into our infrastructure to enhance security. Under PKI, every agent factory possesses a pair of keys. One is the private key which is only

known by the factory itself, and the other one is the public key which should be made known publicly. Information encrypted with the private key can only be decrypted by the matched public key, and vice versa. In SAFER, a fresh agent ID is signed (encrypted) with the private key of the agent factory. Then the fresh agent ID and encrypted one are all included in an agent ID package.

When an agent roams to a host and requests certain services, the first task the host should do is to check the identity of the agent, which is called agent authentication. Figure 6 illustrates the process of agent authentication using a Trusted Factory List (TFL). In SAFER, each host maintains a TFL, which contains the IDs and public keys of trusted agent factories. This list is updated periodically to maintain up-to-date information of trusted factories. When a host authenticates an incoming agent, it first extracts the factory ID from the agent ID package. Then the host searches it within the TFL. If this factory ID cannot be found in the list, authentication fails and service requests will be refused. Otherwise, the public key of the corresponding factory is retrieved and then used to decrypt the encrypted agent ID, producing the decrypted agent ID. Finally, the decrypted agent ID is compared with the fresh one. If these IDs coincide, the host can be sure that the identity of this agent is true. Otherwise, the identity of this agent is suspected or it may have been compromised. In this way, it can be sure that only agents fabricated by trusted factories are served, and any hacker or potential attacker can be detected.

The advantages of this mode of agent authorization and authentication are as follows:

- It is uniform in building the identity of an agent.
- It is more feasible for hosts to maintain a TFL than a list of owners or agents, because the number of agent factories is limited, compared with numerous owners and agents.

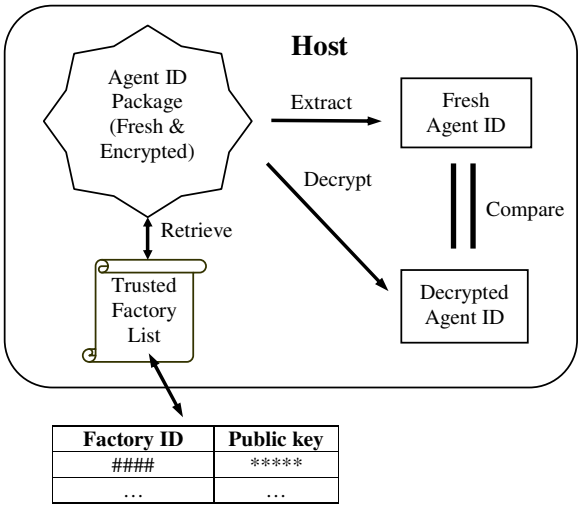

Figure 6. Agent authentication using a Trusted Factory List

Apart from checking the identity of agents, checking the integrity of agent body is also an important procedure in the authentication process. Agent digest, which has been mentioned in section 3.2, can be used for this purpose, while PKI is still a useful tool in protecting the agent digest [Wayner (1995)].

5. Implementation and Discussions

The implementation of agent fabrication is divided into three stages. Firstly, the functions and interfaces of three main facilities in SAFER, namely, CAC, agent factory, and agent owner are designed and implemented. Secondly, a typical agent community, which includes one CAC, one agent factory, and agent owners, is constructed. In this community, formalities of agent fabrication are defined and implemented. Thirdly, an infrastructure of agent authorization and authentication is built into agent fabrication.

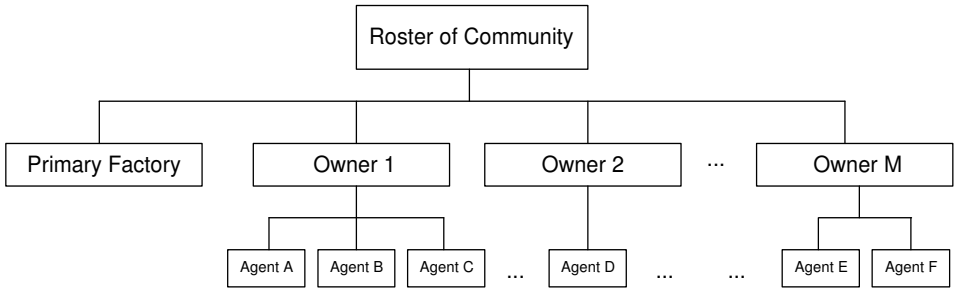

Figure 7. Hierarchical structure of a CAC roster

In a CAC, the most important component is the roster which contains basic information of entities in the community. In order to facilitate the process of updating and retrieving the roster, a hierarchical structure is used to compose the roster, which is shown in Figure 7. There are two layers in the roster. The upper level contains information of registered agent factories and owners, while the lower level contains that of registered agents. Agents are closely attached with their owners in the roster. Therefore, when a new agent is registered in the roster, the roster will try to link it with its owner. In the case that its owner has not registered in the roster, the registration of such an agent will fail. Furthermore, when other entities ask the center to help check the identity of an agent, CAC can complete this task quickly by locating first its owner in the roster. Figure 8 is a screenshot of a CAC. It shows that five owners and one agent factory have been registered in the community. A list is pulled down showing agents which belong to Owner M.

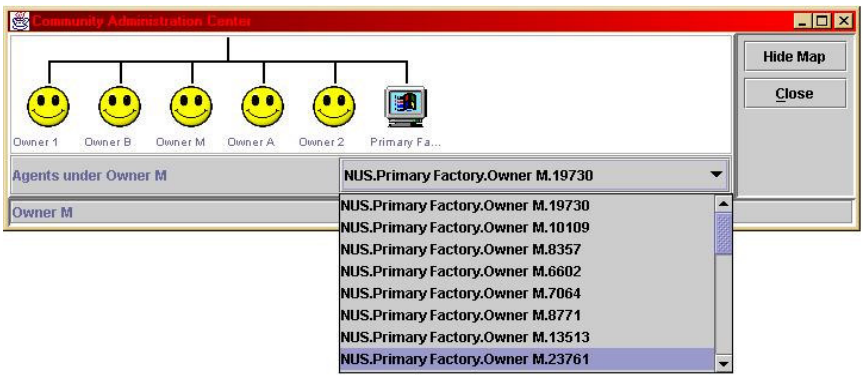

Figure 8. Screenshot of community administration center

The primary task of an agent factory is to fabricate new agents. When it is activated, it waits for incoming requests from agent owners. Till now, four types of agents, namely, brokering agent, negotiation agent, payment agent, and common agent, have been defined in an agent factory. Common agent has the highest flexibility, since it can be customized to possess any combination of functions that the other three types of agents can provide. For instance, a most powerful common agent may embody all the brokering, negotiation, and payment modules. Furthermore, agent factory also provides other functions through

corresponding interfaces such as searching agent records, browsing factory archive, skimming agent catalog, and maintaining the agent factory. The factory archive stores records of all the agents that were fabricated in the factory before. Each agent record in the factory archive includes agentID, ownerID, agent type, etc. Therefore, users can search agent records using the keywords of agent ID, owner ID, and agent type as shown in Figure 9.

Figure 9. Screenshot of agent factory in searching agent records

In order to efficiently control his agents, an agent owner is armed with many user-friendly tools, as shown in Figure 10. As a start, an agent owner should register in CAC through the 'register owner' interface. If he meets the membership criteria, the agent owner will get a certificate together with a successful reply message, and become a member of the community. Then, he can request the agent factory to fabricate a new agent by using the 'request new agent' interface. Figure 10 shows that the agent factory has returned the types of agents that it is able to fabricate, and the owner is prompted to choose one from them. In addition, an owner can check his agents with the 'check agent' interface to examine the module lists of agents or change parameters in some modules. Furthermore, agent owner can also sort the agent list and exchange the modules among agents. If an agent is no longer useful, the owner can delete agents with the 'delete agent' interface.

Figure 10. Screenshot of agent owner requesting to fabricate a new agent

With these three types of entities in place, the formalities of agent fabrication have been implemented. The formalities of agent fabrication have been discussed in section 3.3. Among the three stages of agent fabrication, the customization stage is the most important and complicated. Figure 11 depicts the scenario that an agent owner is customizing a new 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.

Figure 11. Screenshot of a user customizing a new agent

Java is chosen as the language for implementation, because it has important features including robustness, security, and portability. Moreover, Java provides a three-layered security model and many mechanisms to enhance security protection.

Java offers socket-based communication that enables applications to view networking as a simple file I/O. There are two stream-based socket classes: a *ServerSocket* that a server uses to listen for incoming connection and a *Socket* that a client uses in order to initiate a connection. In our implementation, an initiator, e.g. agent owner, creates a Java *Socket* with the agent factory's address and port number, and then initiates the connection. The destination server, e.g. agent factory, creates Java's *ServerSocket* with its port number and listens for incoming connections. Once a connection is established, the data flows between them in continuous streams.

Since an agent factory is expected to deal with many owners simultaneously. Java's multithreading facility is employed to support multiple concurrent subtasks. The basic scheme for handling multiple clients with multithreading is that the agent factory creates a thread attached to Java's *ServerSocket* and listens for incoming connections with *accept()* method. The return value (a *Socket*) is passed to the constructor *Handler*, which creates a new thread to handle that particular connection. Then *accept()* method is called again to wait for a new request from others. Object serialization is another feature provided by Java, which allows an object, that implements the *Serializable* interface, to turn into a sequence of bytes that can be restored fully into the original object later on a different machine. This feature is also utilized to implement message exchanges among entities in the community.

6. Prototype Applications

Two prototype applications have been built to test the proposed agent fabrication scheme. One demonstrates the fabrication of product-brokering agents, and another one demonstrates the fabrication of air-ticket purchasing agents, accomplishing tasks in a virtual marketplace. Both are implemented with the Java language. As the implementation of the SAFER architecture and its entities has been presented in the previous section, we focus on their specific functionalities and fabrication processes here.

The first application built is to fabricate agents that could search for product information in related databases. The agents are expected to accept queries from users, search the corresponding databases, and present results to the users. As a simple testing, the agent factory maintains three types of modules, i.e. SQLquery, sorting, and report. The functionality

of each module is indicated by its name. While a user fabricates a new agent, he/she is prompted with the choice of these three modules to decide which of them will be assembled into the agent body. With different selections, the resulting agents will possess different combinations of the functionalities. This simple application has shown our concepts of agent suitcase and modularization are feasible.

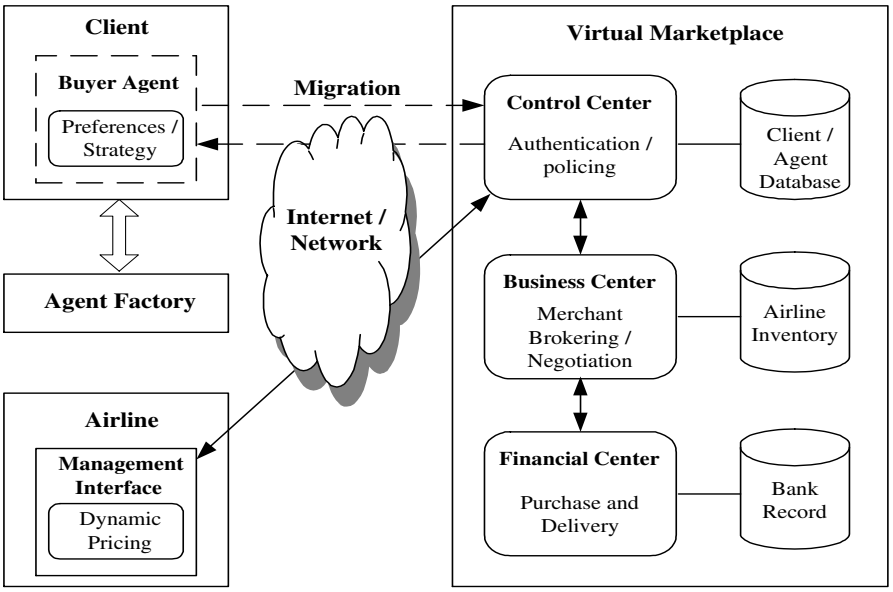

Figure 12. Virtual marketplace architecture

Another application is to fabricate agents that are able to buy air-tickets in a virtual marketplace. This is a relatively complicated application. Figure 12 shows the architecture of the virtual marketplace. It consists of three separate elements, namely, control center, business center, and financial center, which undertaking different functions as shown in the figure. Seller agents are permanent entities residing in the marketplace and they belong to individual airlines, and airline companies can manage their agents via a management interface. 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. The crucial thing here is the fabrication of the buyer agents, especially the process of customization. During fabrication, a user should select 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 the initial offer price, the maximum allowable price, and a choice of three time-based price-adjustment functions. After the agent is successfully fabricated, 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 the 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. Thus, a full air-ticket purchasing process is realized.

The two prototype applications have demonstrated the feasibility of our agent fabrication scheme, although they are still under further development. Some lessons are learnt from the experience of building these applications. For example, the construction and deployment of agent factories require some significant starting effort, and the structure of agents should be less complicated and easy to be modularized.

7. Conclusion

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. A conceptual structure is proposed as a model for e-commerce agents. On the basis of this structure, agent module suitcase is designed and the formalities of agent fabrication are elaborated. Our implementation shows that, with these facilities, agents can be successfully fabricated according to the formalities prescribed and customizations from owners, although these agents are simple with little intelligence. Finally, a new infrastructure of agent authorization and authentication is integrated into agent fabrication, coupling with the PKI technology. The analysis and implementation show that it can enhance the security of agent-based e-commerce.

For future work, our schemes and implementation will be improved in several aspects. Firstly, in order to equip agent factories with the ability of automatic maintenance for agent definitions, an ontology structure has been proposed. Secondly, more flexibility will be added into the stages of agent fabrication. Lastly, a more challenging future work item is regarding how to fabricate self-organizing agents for e-commerce applications.

Acknowledgement

The authors would like to thank MinThein Maung, ChuenHwee Ng, and Sunny Kusnadi for their contributions in implementing the agent fabrication scheme and prototype applications.

Deleted:

References

Chavez, A. and Maes, P. (1998). 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, 75-90.

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

Concordia project, Mitsubishi Electric Information Technology Center America, *http://www.meitca.com/HSL/Projects/Concordia/MobileAgentsWhitePaper.html.*

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

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*, vol. 11(4), 509-525.

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

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

Guan, S.U., Zhu, F.M., and Ko, C.C. (2000). 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, 528-534.

Guttman, R.H. and Maes, P. (1999). Agent-mediated negotiation for retail electronic commerce, in *Agent Mediated Electronic Commerce: First International Workshop on Agent Mediated Electronic Trading*, (Noriega, P., and Sierra, C. ed.), Springer, Berlin, 70-90.

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

Lange, D.B. and Oshima, M. (1998). *Programming and Deploying Mobile Agents with Java Aglets*, Addison-Wesley, Mass., USA.

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

Jardin, C.A. (1997). *Java electronic commerce sourcebook*, Wiley Computer Publishing, New York.

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

Maes, P. (1994). Agents that reduce work and information overload. *Communication of the ACM*, vol. 37(7), 31-40.

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

Poh, T.K. and Guan, S.U. (2000). Internet-enabled smart card agent environment and applications, in *Electronic Commerce: Opportunities and Challenges*, (Rahman, S.M. and Raisinghani, M. ed.), Idea Group, PA, 246-260.

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

Tenenbaum, J.M., Chowdhry, T.S., and Hughes, K. (1997). Eco system: an Internet commerce architecture. *IEEE Computer*, vol. 30(5), 48-55.

Tsvetovatyy, M., Mobasher, B., Gini, M., and Wieckowski, Z. (1997). MAGMA: an agent based virtual market for electronic commerce. *Applied Artificial Intelligence,* vol. 11(6), 501-524.

Wang, T.H., Guan, S.U., and Chan, T.K. (2001). Integrity protection for code-on-demand mobile agents in e-commerce, to appear in *Special Issue of Journal of Systems and Software*.

Wayner, P. (1995). *Agent Unleashed: A Public Domain Look at Agent Technology,* Academic Press, London.

Wong, D., Paciorek, N., Walsh, T., et al. (1997). Concordia: an infrastructure for collaborating mobile agents, in *Proceedings of First International Workshop on Mobile Agents,* Berlin, Germany.

Wurman, P.R., Wellman, M.P., and Walsh, W.E. (1998). 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, 301-308.

Yang, Y. and Guan, S.U. (2000). Intelligent mobile agents for e-commerce: security issues and agent transport, in *Electronic Commerce: Opportunities and Challenges*, (Rahman, S.M. and Raisinghani, M. ed.). Idea Group, PA, 321-336.

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

Zhu, F.M. and Guan, S.U. (2001). Towards evolution of software agents in electronic commerce, in *Proceedings of the IEEE Congress on Evolutionary Computation 2001*, Seoul, Korea, 1303-1308.