Integrity Protection for Code-on-Demand Mobile Agents in E-Commerce

Tianhan Wang, Sheng-Uei Guan*, and Tai Khoon Chan

Department of Electrical and Computer Engineering

National University of Singapore

10 Kent Ridge Crescent, Singapore 119260

Email: {engp9592,eleguans,eng81022}@nus.edu.sg

* Corresponding Author

Abstract

The mobile agent paradigm has been proposed as a promising solution to facilitate distributed computing over open and heterogeneous networks. Mobility, autonomy, and intelligence are identified as key features of mobile agent systems and enabling characteristics for the next-generation smart electronic commerce on the Internet. However, security-related issues, especially integrity protection in mobile agent technology, still hinder the widespread use of software agents: from the agent's perspective, mobile agent integrity should be protected against attacks from malicious hosts and other agents. In this paper, we present Code-on-Demand(CoD) mobile agents and a corresponding agent integrity protection scheme. Compared to the traditional assumption that mobile agents consist of invariant code parts, we propose the use of dynamically upgradeable agent code, in which new agent function modules can be added and redundant ones can be deleted at runtime. This approach will reduce the weight of agent programs, equip mobile agents with more flexibility, enhance code privacy and help the recoverability of agents after attack. In order to meet the security challenges for agent integrity protection, we propose agent code change authorization protocols and a double integrity verification scheme. Finally, we discuss the Java implementation of CoD mobile agents and integrity protection.

Keywords: Mobile agents; Integrity protection; Code-on-Demand; Electronic commerce

1. Introduction

Electronic commerce is to use electronic means for conducting business transactions. Its implementation harnesses networked resources to foster the exchange of business transactions in a more

efficient and cost effective manner (Aaron, 1999). The market of e-commerce is growing very rapidly thanks to the increasingly greater accessibility of Internet services. The World Trade Organization expects worldwide E-commerce revenues to reach two hundred billion US dollars in the next two years. E-commerce brings benefits to both suppliers and customers.

Besides current web-based systems, agent-mediated e-commerce is now playing a more and more important role. Mobile agents refer to self-contained and identifiable computer programs that can move within the network and act on behalf of the user or another entity (Pham and Karmouch, 1998). The mobile agent paradigm has been proposed as a competitive counterpart to the client/server paradigm. Instead of exchanging messages known as request/response between the client and server, mobile agents can migrate to a remote host and execute, thus taking advantages of exploiting the resource near the data source and reducing network traffics caused by frequent remote communications. Mobile agents may be equipped with intelligence ranging from simple decision algorithms to sophisticated reasoning and learning.

Mobile agents have been proposed in many agent-based e-commerce systems and architectures (Greenberg et al., 1998) such as information gathering, comparison-shopping, agent-based auctions (Wang et al., 2000)(Wurman et al., 1998), and agent-based payment systems (Hua and Guan, 2000), etc. From the agent user's perspective, mobile agents serve as "personal software assistants" acting on behalf of the owner in completing e-commerce related tasks. The mobility, autonomy, and intelligence of mobile agents will be the key elements in providing solutions for a flexible and smart e-commerce era of the next generation.

In agent-mediated e-commerce systems, one of the biggest concerns arises from security related issues, especially the integrity protection of mobile agents when they are dispatched into open and

heterogeneous networks. Comparing the protection of network hosts with the protection of mobile agents (Marques et al., 1999), it is safe to say that protecting mobile agents is much more difficult than protecting hosts, considering the fact that agent programs have to rely on the host platforms to execute them. Most research carried out in this area typically makes the assumption that the code of agent programs is invariant after construction. This assumption has virtually made the integrity protection of agent code much easier because traditional methods based on digital signatures and hash functions can be used to verify the authenticity and integrity of mobile agent code.

What are the benefits if agent code is dynamically changeable? In this paper, we propose mobile agents with a Code-on-Demand feature: a mobile agent can have its code upgraded with new function modules added or redundant ones deleted depending on the executing requirement. This type of agents will prove to be more flexible in e-commerce compared to the traditional static type. The challenges brought up by our CoD mobile agents are obvious: agent integrity should still be verified and ensured even if code changes.

This paper is organized as follows: section 2 reviews some previous research work in the direction of protecting mobile agents, assuming the static code model. Section 3 introduces CoD mobile agents and discusses advantages to their static counterpart. Section 4 discusses code change authorization protocols and proposes an integrity protection scheme based on double integrity verification. Laboratory prototype implementation is discussed in details in section 5. In the end, we conclude this paper and look into future research work.

2. Related work in agent integrity protection

Mobile agents become especially vulnerable when traveling among network hosts. For a reliable e-commerce system, mobile agents must be protected from attacks from malicious hosts and agents. There are generally two directions in protecting mobile agents: detection and prevention (Vigna, 1998). Detection is aimed at detecting any malicious attack after it has happened. This can be helpful to immediately adopt remedial procedures after specific attacks have been identified. Prevention schemes are aimed at preventing the compromise of mobile agents in a specific way. Most of the work assumes that the code part of a mobile agent is invariant and what is changeable is only the agent executing state and the data blocks collected at each foreign host.

2.1 Detection

Schemes for detecting compromise of mobile agents treat code and data differently. Code integrity is treated in a simple way because they can be verified using the traditional cryptographic ways such as the digital signature over the agent code digest. The verification can be done using trusted hosts or collaborative agents.

For the dynamically changeable data part, various approaches have been developed. Multiple Hops (MH) (Corradi et al., 1999a)(Corradi et al., 1999b) proposed by A. Corradi *et al*. is an integrity protection scheme for the SOMA project. It uses a shared secret stored in the original sender to detect any intermediate modifications when a mobile agent is visiting a series of hosts. This may provide some clues to deal with any change in the code part. However, one of the drawbacks of this approach is that the integrity verification can only be done after the mobile agent has returned to the original sender. This clearly is not suitable for the case of code part, because agent code should be verified intermediately to ensure its faithful execution. There are also some potential security loopholes in this approach, such as

the “visit-once” assumption made by the authors and the problem arising if a few hosts collaborate to cheat together.

2.2 Prevention

While most detection schemes focus on dynamically changeable agent data or state, prevention mechanisms try to make impossible any specific modification to the code part - which is static. Code cannot be prevented from modification since it is running on a foreign machine and thus is at the mercy of that host. What we can do is to prevent a “specific” modification that will be favorable to the host’s advantage. For example, in agent-based comparison-shopping, an agent is sent out to visit a series of venders’ hosts that sell traveling tickets. The later visited vender host may analyze the agent code and make modifications to its advantages: a malicious host may retrieve the information on the highest price the agent owner can accept and then offer a price better than the offers made by the previous hosts and below the highest price acceptable by the owner.

One solution is to employ tamper-resistant hardware to execute agents in a physically sealed environment. However, these devices are relatively expensive. There are also some software-based approaches. A black box security mechanism (Hohl, 1998) proposes to create a black box out of an original agent. A black box is an agent that performs the same work as the original agent, yet with a different structure. Function hiding (Sander and Tschudin, 1998) is a software protection scheme applied to protecting mobile agents against malicious hosts. This approach hides important functions to hosts and makes real functions in the original code illegible. However, this has only been applied to specific cases for rational and polynomial functions.

The assumption that agent code is invariant is made in most research work on integrity protection. In this case, the integrity of static mobile code can be verified using the following procedures: (1) construct a mobile agent with static code. (2) use a one-way hash function to compute a digest over the agent code. (3) use the agent code provider's private key to digitally sign this digest. The verification can be done reverse: first check whether the hash value has a valid digital signature using the agent provider's public key, assuming that a public key directory service has already been established. If the digital signature can be verified successfully, a further step may be taken to compute the agent code using the same hash function and evaluate this value against the original digest provided. If it is not the same, a conclusion can be drawn that agent code is compromised.

In this paper, we go one step further by removing the assumption that agent code is static. We propose CoD mobile agents whose code parts are changeable at runtime. This will inevitably bring potential security challenges on agent protection, especially on code integrity. Our goals in integrity protection are:

(1) Allow agent code to change in an authorized fashion.

(2) The validity of changeable code can be checked to ensure the integrity of agent code.

(3) Host can also verify agent code to be assured of its validity.

3. CoD mobile agents

3.1 SAFER framework

We are considering a mobile agent community proposed as in SAFER (Zhu et al., 2000)(Guan and Yang, 1999)(Yang and Guan, 2000): **S**ecure **A**gent **F**abrication, **E**volution and **R**oaming. SAFER

provides an infrastructure for intelligent agent-mediated electronic commerce. The goal of SAFER is to construct a secure, standard, and evolutionary agent system for electronic commerce based applications and transactions. Figure 1 briefly sketches such a SAFER agent community.

{Insert Figure 1 here}

Each SAFER community comprises various components and entities, such as the agent butler, agent factory, community administration center, etc. We won't discuss the details since they can be found in the earlier papers in (Zhu et al., 2000)(Guan and Yang, 1999)(Yang and Guan, 2000). For the clarity of later sections, we briefly introduce several entities relevant to our work. The agent butler acts on behalf of the agent owner for operating various mobile agents. The agent factory is responsible for fabricating mobile agents and other related code modules. Codes provided by the agent factory should bear the original signature of the factory to provide a means to verify the authenticity of the agent code. TTP is a SAFER certified trusted party in which CoD can be performed.

We have integrated our CoD mobile agents and security measures into the SAFER environment. We start by describing the ideas on CoD mobile agents and then the corresponding integrity protection scheme.

3.2 What are CoD mobile agents

In the following sections, we introduce the basic concepts of CoD mobile agents. Let's do so by citing the example of comparison-shopping in Sec. 2.2. Assume mobile agent is sent out to visit a few vendor hosts to collect information on a few products and make decisions on each based on some complex decision algorithms. These decision algorithms are different and specific to each product. After the agent has visited all the hosts for a certain product and collected information enough to make a final

decision, the corresponding algorithm in the code module is used. Should this agent carry all the decision function modules when sent out by the agent owner, as is normally done in the static agent code case before?

Obviously, this is not efficient, since the agent only needs specific modules when making a decision on a certain product. Carrying all function modules at startup obviously has the following disadvantages:

(1) It increases the "body weight" of mobile agents. This will add additional time/cost for agent transportation and add burdens to the limited network bandwidth. The advantages of reducing client/server communication is traded with agent transportation time/cost and agent weight.

(2) It exposes more sensitive function modules to threats, since a mobile agent has to carry all the function modules regardless whether or not it needs them immediately.

We propose CoD mobile agents as a potential solution to the problems. A CoD mobile agent is an agent whose code is programmed such that function modules can be dynamically added or deleted depending on the agent executing status. The agent can be upgraded to include some new code part to function. We further elaborate these two kinds of changes that can be made on agent code.

- Addition: Agent function modules can be dynamically added to the existing agent code to form an upgraded version. Each function module should include the function code and also the proper digital certificate regarding from which this code is fabricated, namely, the source agent factory as a proof of its validity. For example, a digital signature of the agent factory over the collision resistant hash value of the function module can be helpful to prove both the authenticity and the integrity of the code. Addition of any particular code modules should get authorization from the proper parties.

- Deletion: Agent function modules can be dynamically deleted from the agent body to form an upgraded version. Deletion of function modules should also get authorization from the proper parties.

Figure 2 depicts the functioning of a CoD mobile agent. This figure includes the SAFER components of the agent factory, agent butler, TTP and a few network hosts the agent is to visit. The agent butler, acting on behalf of the user, may download mobile agents from the agent factory, and dispatch them to remote hosts. During the period when a mobile agent is traveling among hosts, its structure including code and data part may be subject to changes in response to tasks carried. This means new data blocks and Agent Function Modules (AFM) can be added and existing ones can be deleted. However, such changes must follow the code change authorization protocol (elaborated in Section 4.1) depicted in the left part of the figure with the interaction of SAFER components of the agent factory, agent butler and TTP.

{Insert Figure 2 here}

For example, after the original agent provided by the factory is sent out by the user, it travels to a series of hosts to complete some tasks. When it finds out it needs some additional modules, it will seek to download these modules from the code provider – the agent factory. In host 2, both a new function module and data blocks are appended to the agent body. Similarly, when the agent finds that it does not need certain redundant modules, it will discard these modules in order to keep itself light-weighted.

3.3 Why Code-on-Demand

Compared to the static-code mobile agent which is generally assumed, our CoD mobile agent exhibits some noticeable advantages:

- As mentioned above, CoD feature reduces the cost caused by carrying code modules not needed in the near future.
- Code privacy is enhanced. Since code module will only be added when needed and deleted when they become useless, irrelevant hosts will be less likely to spy on the important algorithms from the decision modules. In this case, the threat that agent code will suffer from malicious tampering is reduced.
- CoD will improve the flexibility of agent functionality. Dynamic code will enable a mobile agent to process information that is originally inaccessible. The agent can even change a particular function module if needed. For example, a user may want his mobile agent participating in an auction to have different bidding strategies to use when bidding for different products.
- In case that a malicious attack on agent code really happens and has been detected, CoD will be helpful to recover a compromised agent. Such a mobile agent may still function correctly after being examined and replaced of the altered code module.

Although CoD brings more advantages and makes the agent more flexible, it also incurs additional security challenges, especially when looking into the issue of verifying agent code integrity after the code as a whole has been changed.

4. Integrity protection for CoD mobile agents

The verification of agent integrity consists of verifying different parts of an agent. When a mobile agent migrates in an open network environment, this becomes particularly important. As for the agent code part, integrity protection has at least two meaningful purposes. From the agent's perspective, the verification will detect whether or not the agent code has been modified. If any compromise exists, it should be detected. From the host's perspective, the host would also need a proper mechanism be devised for it to verify the validity of an incoming agent.

When the agent code as a whole is dynamically changeable, this verification task becomes complicated. If we still use the simplified methods by validating only the digital signature on the digest from the code provider, the verification will fail since agent code has already been changed to include new or remove existing code modules. The host may mistakenly reject a mobile agent and identify that the agent has been compromised. Currently, we solve this problem by employing a supervised authorization protocol to avoid any unauthorized additions/deletions taken on the agent code. We also use double verification to ensure that the integrity of agent code modules both as individuals and as a whole can be verified successfully.

4.1 Authorization protocols

4.1.1 Supervised authorization protocol

A supervised authorization is needed when more secured addition/deletion of agent code is needed. The supervised authorization involves the agent butler, agent factory and TTP. The update of agent code happens on a TTP.

In the supervised authorization protocol, both code addition and deletion should seek authorizations from the agent butler (on behalf of the agent owner in SAFER). In supervised addition as illustrated in Figure 3, an authorization proof will be issued by the code provider, namely the agent factory, and then a permit is forwarded to the mobile agent by the agent butler.

{Insert Figure 3 here}

The mobile agent needs to send a code change {addition, deletion} request to the agent butler before moving to a TTP for code upgrade.

Request: AGT → USR: Permit Request: {AGT_ID, Host_ID, Change_Type, AFM_ID, AFM_Version, Download_Type}

The agent butler will verify if the request comes from a valid agent that he sent out and identify the change type. If it is a deletion request, the butler can send the authorization permit if it is approved. Otherwise it will forward the addition request to the agent factory:

Request: USR → FAC: Proof Request: {USR_ID, AGT_ID, Change_Type, AFM_ID, AFM_Version, Download_Type}

The agent factory will verify if this request is valid. The factory will also check if this particular AFM (Agent Function Module) is available for downloading. “Download_Type” means that AFMs can be either downloaded from the agent factory, or from local cache on the TTP if available and the digital signature and integrity can be successfully verified. We will allow the addition of AFM directly from the database but even in this case the authorization permit must be received and verified before addition can take place. If the factory has a newer version of AFM available than the one in the TTP’s database, it

will either provide an older version or substitute with the latest version, depending on the agent butler's discretion.

The factory shall send an authorization proof to the agent butler informing whether the code change request is approved or not. The agent butler will forward the permit message to the mobile agent. After obtaining authorization, the agent can start adding a new function module either locally or from the remote agent factory, or deleting an existing module.

In the supervised authorization protocol, the agent butler is involved and will keep a record on the changes of agent code. This is for the purpose of providing up-to-date agent code status to hosts enquiries in later stages.

4.1.2 Unsupervised authorization protocol

In the unsupervised authorization protocol, the agent butler is not directly involved in the authorization process but will be informed after the authorization between the agent and the factory has been done. The agent can be equipped with the intelligence to decide whether to communicate with the agent factory directly for adding a certain function module or to delete useless code modules. If the agent is roaming in a relatively trusted host community, this approach will improve efficiency but obviously with less security: the agent butler may lose track of its agent code status eventually.

4.2 Double integrity verification

Integrity verification actually has two purposes. To hosts, before any agent can execute on the host environment, it must be thoroughly examined by the host. As long as agent code has undergone valid changes, such as authorized addition and deletion, there is no reason that the host should not accept

a CoD mobile agent. On the other hand, to the mobile agent, any compromise on the code must be detected.

Here we would like to differentiate the integrity verification of a single code module and that of the agent code as a whole. An agent can be deleted of any code module entirely without affecting the integrity of the other code modules. So both individual and overall code integrity should be verified.

We use the combination of a one-way collision-resistant hash function and digital signature to form a double verification scheme. For each code module fabricated by the agent factory, a digest using the hash function is computed, denoted as *IMD*. This digest is digitally signed using the private key of the agent factory, denoted as $S_{FAC}(IMD)$. Another digest of the overall agent code, including the original agent body, together with all the currently added code modules contribute to the value of *OMD*. Each time the agent roams to a TTP to have its code updated, *OMD* will also be updated. This digest is further signed digitally by the trusted host using its own private key denoted as $S_{TTP}(OMD)$.

The task of integrity verification involves double verification of the digital signature and hash values of both *IMD* and *OMD*. The verification of digital signatures indicates whether or not the hash value is valid from the computing party of the value. *IMD* verification indicates whether the individual code part has been compromised; while *OMD* indicates the integrity of agent code as a whole.

4.3 Discussions

We expect that CoD mobile agents roam among SAFER-compliant communities when executing e-commerce tasks, thus necessary entities mentioned above are available in each SAFER community. The entry or leave of mobile agents to each community is assisted by the “Immigration” interface of a SAFER community. In order for mobile agents to find the most suitable resource server (such as TTP, agent factory), we assume that a global directory service that stores information for all registered SAFER

entities has been established. A “global” service stands above all registered SAFER communities compared to “local” service which is available only within a certain SAFER community. With this assistance, our CoD agents are able to use global IDs to address and identify a target destination (such as a global agent factory ID) or target resource (such as a global AFM ID) most efficiently. For example, demanded AFMs may be downloaded from the closest agent factory available, but not necessarily from the original one.

There are also alternatives that we may consider for designing dynamically coded mobile agents. For example, Agent-on-Demand which allows the whole mobile agent available for downloading from a remote host, may enhance the trustablity and coherence of the agent code as a whole, as sometimes the introduction of partial code modules may cause unexpected problems.

5. Prototype implementation

5.1 System overview

Using Java as the language for development, we have built a prototype to investigate and demonstrate the possibility of CoD mobile agents. The system is developed in Java due to the fact that Java provides cross-platform portability.

Java also provides dynamic extension (Venners, 1998) possibility through the usage of forName() and ClassLoader. In the prototype, we used ClassLoader to load both agent classes and AFM. The ability to load AFM dynamically is what we call Code-on-Demand. We have also implemented the supervised authorization protocol and double integrity verification to protect the integrity of the agent and its AFM. We have used MD-5, a hash algorithm that produces 128-bit hash values to digitally hash

agent codes in the prototype. The resulting hash values are then digitally signed using Digital Signature Algorithm (DSA).

5.2 Dynamic extension

The Java architecture allows for dynamic extension, which means that a Java program can dynamically load and use classes at runtime. In other words, a Java program does not necessarily need to know all the classes at compile time. Thus, dynamic extension has the flexibility to allow a program to download necessary function modules when needed.

Java provides two ways for dynamic extension: forName( ), and ClassLoader. A straightforward method is to use forName( ) to implement dynamic extension. ClassLoader, on the other hand provides a more powerful and secure approach to load classes at runtime. ClassLoader written in our prototype is a subclass of java.lang.ClassLoader.

ClassLoader provides the ability to load classes into different namespaces and thus shielding types from each other. In this way, agents or codes loaded into one name space will not be able to meddle with agents from other name spaces. Moreover, ClassLoader allows for live replacement such that a new class object with the same name can replace that particular class object. The old class object can easily be thrown away by dropping all references to it. This is not possible with forName( ).

5.3 Implementation of CoD

We have developed a car dealer application in which we use a mobile agent to represent a buyer seeking a new car. The agent moves from one vendor host to another looking for cars that meet the requirements. Our agent uses JDBC to access MySql database in the search for a suitable car. However, when the requirements such as price, capacity are similar, the agent will employ a unique algorithm to

decide on the best buy. In our example, when such a situation arises in which a unique complex decision algorithm is required, the agent will first check whether it has the code. If some functions are seldom used, the lightweight agent will not be carrying the extra function modules. In this case, the agent will employ our code change authorization protocol to dynamically load the required function module. The following code demonstrates how we use the Java ClassLoader to load function modules dynamically.

```
CoDClassLoader cod = new CoDClassLoader();
 try {
        // Load Function_A_Class
       Class Function_A = cod.loadClass(Function_A_Class);
       // Instantiate it as an object
       Object Fn_obj = Function_A.newInstance();
       // Cast the Object ref to the COD Function interface type
      CoD_Function codload = (CoD_Function) Fn_obj;
       // Activate the downloaded code function
      codload.codmain();
} catch (Exception e) {
     System.out.println("CoD class not found!! Terminate CoD loading!!");
}
```

As shown in the code, agent application first instantiates a CoDClassLoader. The resulting object will be a ClassLoader object because class CoDClassLoader extends class java.lang.ClassLoader. The agent then invokes loadClass() on this ClassLoader object, passing it the AFM's name. In this case, the AFM's name is Function_A_Class. loadClass() will attempt to load AFM into its name space. If it is successful, loadClass() will return a reference to the Class instance that represents the newly loaded type. The agent will then be able to invoke function in the Function_A_Class object. In Java 1.2, concrete default implementation of loadClass() is added to java.lang.ClassLoader and the preferred approach to write a ClassLoader in Java 1.2 (Venners, 1998) is to implement the findclass() method. Therefore, we have created our own findclass() method in our prototype to dynamically load required a AFM. An

exception will be generated when the CoDClassloader is not able to find the Function_A_Class. The agent will then identify the missing function module by extracting the error code from the exception.

Likewise, AgentClassloader is also implemented in the same way. Figure 4 shows the relationships between the ClassLoader objects and their namespaces. Namespace is the part of memory area that is private to the ClassLoader. As we can see from Figure 4, the HostClass and the AgentClassLoader are both loaded by the Primordial ClassLoader. Primordial ClassLoader which is part of the JVM implementation is sometimes called the system ClassLoader or the default ClassLoader. Our AgentClassLoader after being loaded into memory is in turn used to load the CoDClassLoader and the buyer agent. We can see from the figure that the CoDClassLoader and the buyer agent both sit in the namespace of the AgentClassLoader. Subsequently, CoDClassLoader when invoke, will load Function_A_Class and any other CoD classes into its own namespace.

{Insert Figure 4 here}

After the exception is generated, the agent will terminate and the host will request for code addition authorization from the agent butler as described in the design. After an authorization permit is obtained, the agent and the permit will then be transported to a TTP. TTP will then use the permit to request for the new function module from the agent factory. Agent factory on receiving the permit will verify the validity and issue the new function module accordingly. The new AFM will be sent to TTP where double integrity verification will be used to protect the integrity of both the agent code and its AFM. Subsequently, the updated agent is then sent back to the original vendor host. The agent equipped with an updated decision algorithm will be able to make decision on the car to purchase.

Figure 5 shows the vendor host's user interface in our prototype system. As seen from the screen shot, there are three text boxes, which shows agent action, security check status and communication

status respectively. The figure illustrates the situation when a requested AFM is not found. Figure 6 shows the screen display of the agent butler receiving an authorization permit.

{Insert Figure 5 here}
{Insert Figure 6 here}

5.4 Integrity protection

We have applied double integrity verification technique to our prototype system as shown in figure 4. Firstly the agent factory digitally signs the AFM code. A hash algorithm MD-5 is used to produce a 128-bit hash value of the agent function module. The resulting hash value is then digitally signed using DSA with the agent factory's 1024 bits private key. The resulting signature file and the agent function module are then sent to TTP.

TTP upon receiving the agent function module will produce a hash value of the received code and verify it by using an instance of the Signature class. A Signature object is created using the same signature algorithm as was used to generate the signature. After verifying the integrity of AFM, TTP will then digitally sign the updated agent with its own private key. This is what we called double integrity veryfication by authenticating both individual and overall code integrity. Figure 7 shows the screen display when the TTP generates signature for both the agent and its AFM. Finally, the updated buyer agent is then sent back to the vendor host. Figure 8 shows an AFM being loaded successfully by the buyer agent.

{Insert Figure 7 here}
{Insert Figure 8 here}

6. Conclusions

In this paper, we have proposed a novel type of CoD mobile agents. CoD agents are mobile agents whose code is dynamically upgradeable. This feature is attractive in improving the flexibility of mobile agents and may have much potential usage in the rapidly developing agent-mediated electronic commerce applications. Besides, CoD can help to protect code privacy, reduce transport cost, and help in agent recovery after malicious attacks. Regarding the challenges brought up by code variation, we provide security schemes of code change authorization and double integrity verification. Through the implementation of a practical solution, it has shown that CoD is feasible and security protection can be achieved with Java's advanced feature of dynamic extension.

We have addressed CoD mobile agents and security issues in electronic commerce scenarios. Mobile agents as flexible, autonomous and intelligent entities have made them a viable solution to be employed in e-commerce applications. However, security problems such as integrity protection must be resolved before they come into full commercial use. We do not claim that we have provided a complete solution to this new type of dynamic code mobile agents; rather, we treat this as an attractive feature in SAFER framework and a first step toward deeper research in this direction.

References

Aaron, R., Decina, M., and Skillen, R., 1999. Electronic Commerce: Enablers and Implications. IEEE Communications Magazine, Volume 37 (9), pp. 47 –52.

Corradi, A., Montanari, R., and Stefanelli, C., 1999a. Mobile Agents Integrity in E-commerce Applications. Proc. of the 19th IEEE International Conference on Distributed Computing Systems Workshops on Electronic Commerce and Web-based Applications/Middleware, pp.59-64.

Corradi, A., Montanari, R., and Stefanelli, C., 1999b. Security Issues in Mobile Agent Technology. Proc. of the 7th IEEE Workshop on Future Trends of Distributed Computing Systems, pp. 3-8.

Greenberg, M.S., Byington, L.C., and Harper D.G., 1998. Mobile Agents and Security. IEEE Communications Magazine 36 (7), pp.76-85.

Guan, S.U. and Yang Y., 1999. SAFE: Secure-Roaming Agent For E-Commerce. Proc. of the 26th International Conference on Computers & Industrial Engineering, pp. 33-37.

Hohl, F., 1998. Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. In Giovanni Vigna (Ed.): Mobile Agents and Security. Springer-Verlag, pp.92-113.

Hua, F., and Guan, S.U., 2000. An Agent-based Electronic Payment Scheme for E-commerce. In Rahman, S.M. and Bignall, R.J. (Eds.), Internet Commerce and Software Agents: Cases, Technologies and Opportunities, IDEA Group Publishing, pp. 317-330.

Marques, P.J., Silva, L.M., and Silva, J.G., 1999. Security Mechanisms for Using Mobie Agents in Electronic Commerce. Proc. of 18th IEEE Symposium on Reliable Distributed Systems, pp. 378-383.

Pham, V.A. and Karmouch, A., 1998. Mobile Software Agents: An Overview. IEEE Communications Magazine 36 (7), pp.26-37.

Sander, T. and Tschudin, C. F., 1998. Protecting Mobile Agents Against Malicious Hosts. In Vigna, G. (Ed.): Mobile Agents and Security. Springer-Verlag, pp. 44-60.

Venners, B., 1998. Inside the Java Virtual Machine. New York : McGraw-Hill, Ch 8

Vigna, G., 1998. Cryptographic Traces for Mobile Agents. In Vigna, G. (Ed.), Mobile Agents and Security, Spinger-Verlag, pp.137-153.

Wang, T.H., Guan, S.U., and Ong S.H., 2000. An Agent Based Auction Service for Electronic Commerce. Proc. of International ICSC Congress on Intelligent Systems & Applications, CD #1524-045.

Wurman, P.R., Wellman, M.P., and Walsh, W.E., 1998. The Michigan Internet AuctionBot: A Configurable Auction Server for Human and Software Agents. Proc. of the 2nd International Conference on Autonomous Agents, pp. 301-308 .

Yang, Y. and Guan, S.U., 2000. Intelligent Mobile Agents for E-Commerce: Security Issues and Agent Transport. In Rahman, S.M. and Raisinghani, M. (Eds.), Electronic Commerce: Opportunities and Chanllenges. IDEA Group Publishing, pp.321-336

Zhu, F.M., Guan, S.U., and Yang, Y., 2000. SAFER E-Commerce: Secure Agent Fabrication, Evolution & Roaming for E-Commerce. In Rahman, S.M. and Bignall, R.J. (Eds.), Internet Commerce and Software Agents: Cases, Technologies and Opportunities, IDEA Group Publishing, pp. 190-206

Authors' biographies:

Tianhan Wang received the B.S. degree in Electronics and Information Science and Technology from Peking University in 1999. He is currently a graduate student working towards his M.Eng. degree in Electrical and Computer Engineering at the National University of Singapore. His research interests include electronic commerce and computer networks.

Dr. Sheng-Uei Guan received his B.S. in mathematics from Tsing Hua University in 1979 and his M.S. and Ph.D. in computer science from the University of North Carolina at Chapel Hill in 1989. He was an associate professor of the Department of Computer Engineering and Science at Yuan-Ze University, Taiwan from 1992-1996. He was with the School of Computer Science and Computer

Engineering, La Trobe University in Australia from 1996-1998. He is now with the Department of Electrical & Computer Engineering at National University of Singapore. His research interests include electronic commerce, computer networks, multimedia systems, and intelligent systems.

Tai Khoon Chan is currently an undergraduate in electrical & computer engineering at the National University of Singapore. His research interests include electronic commerce, multimedia networking, and computer networks.

Figures

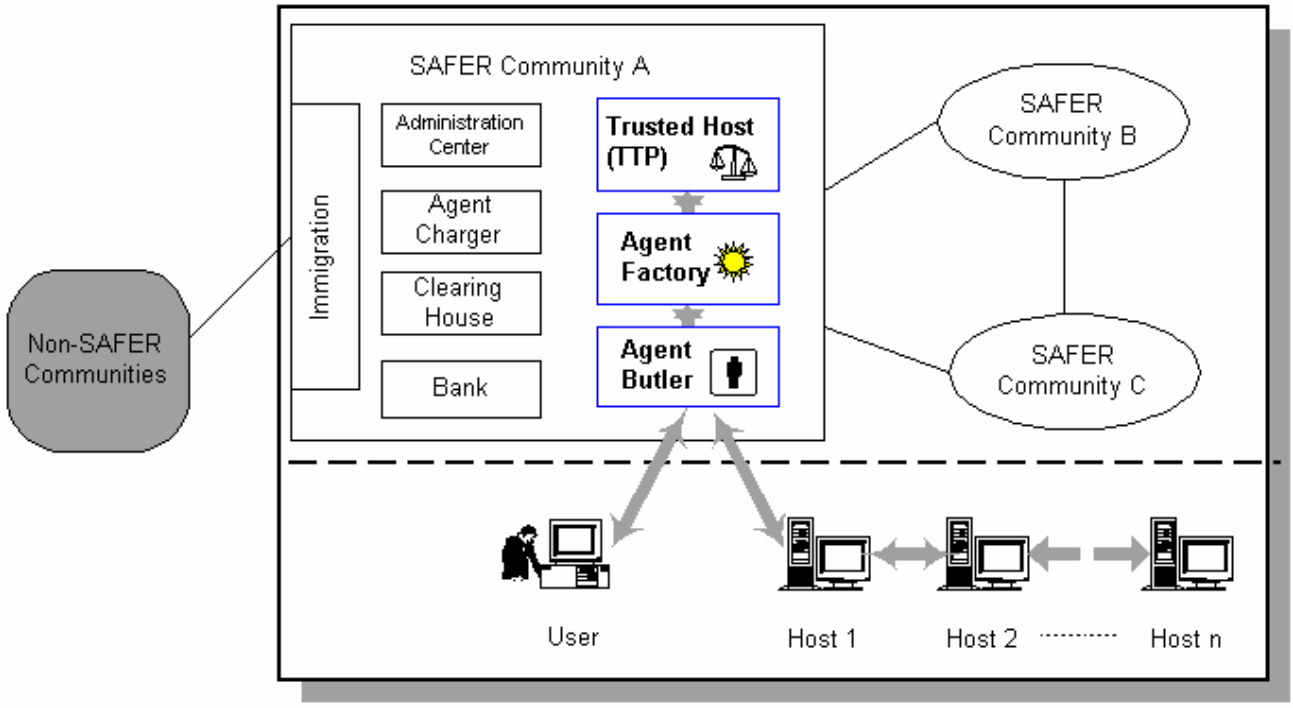

Figure 1. SAFER Agent Community

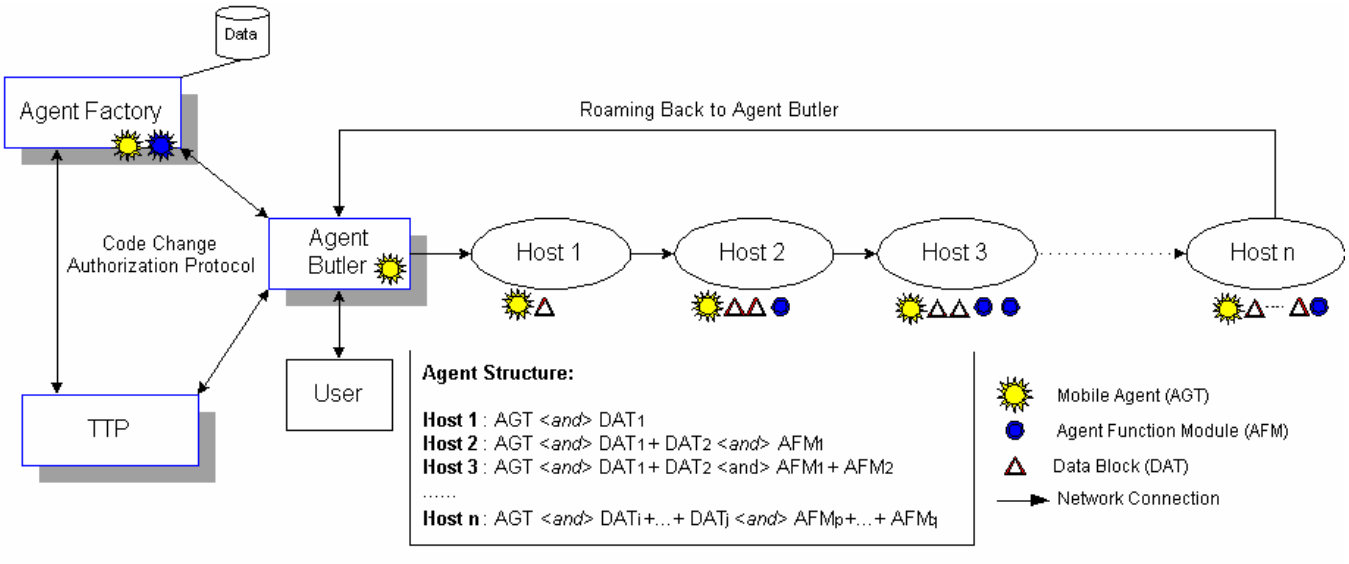

Figure 2. Functioning of a CoD Mobile Agent

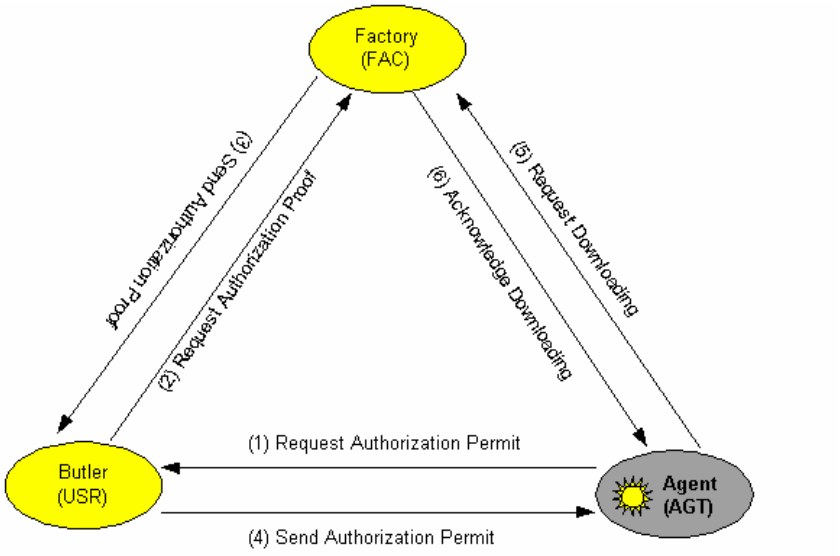

Figure 3. Supervised Authorization Protocol for Code Addition

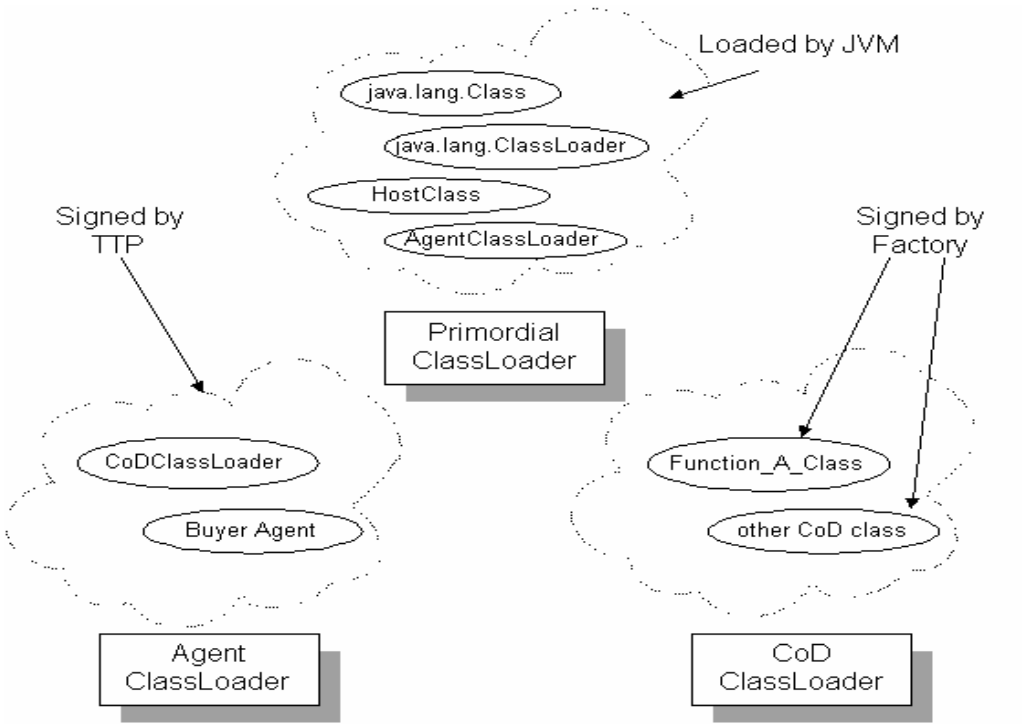

Figure 4. ClassLoader Objects and Name Spaces

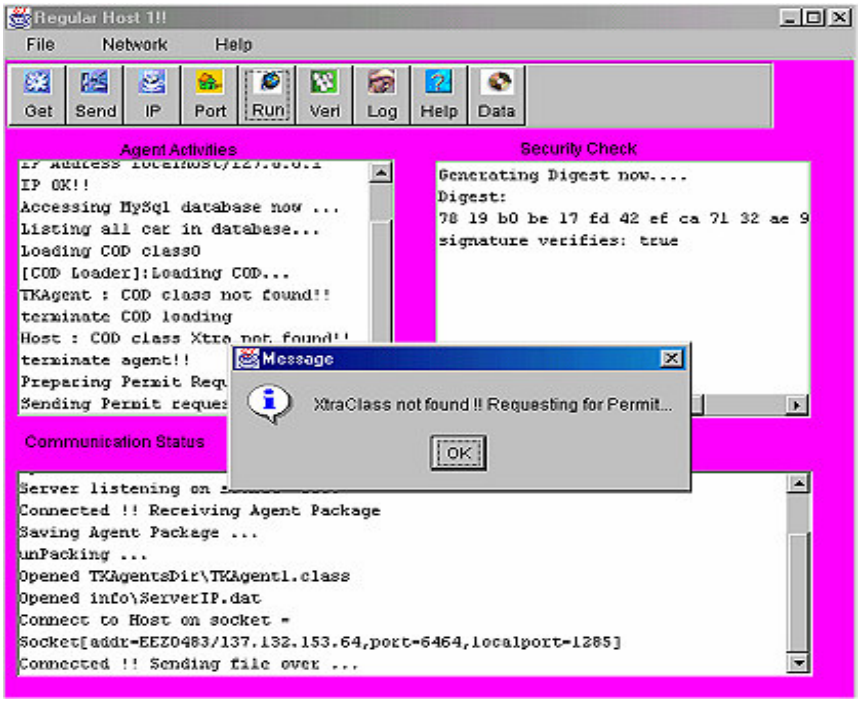

Figure 5. Sample Screen Shot of Vendor Host When Needed AFM Not Found

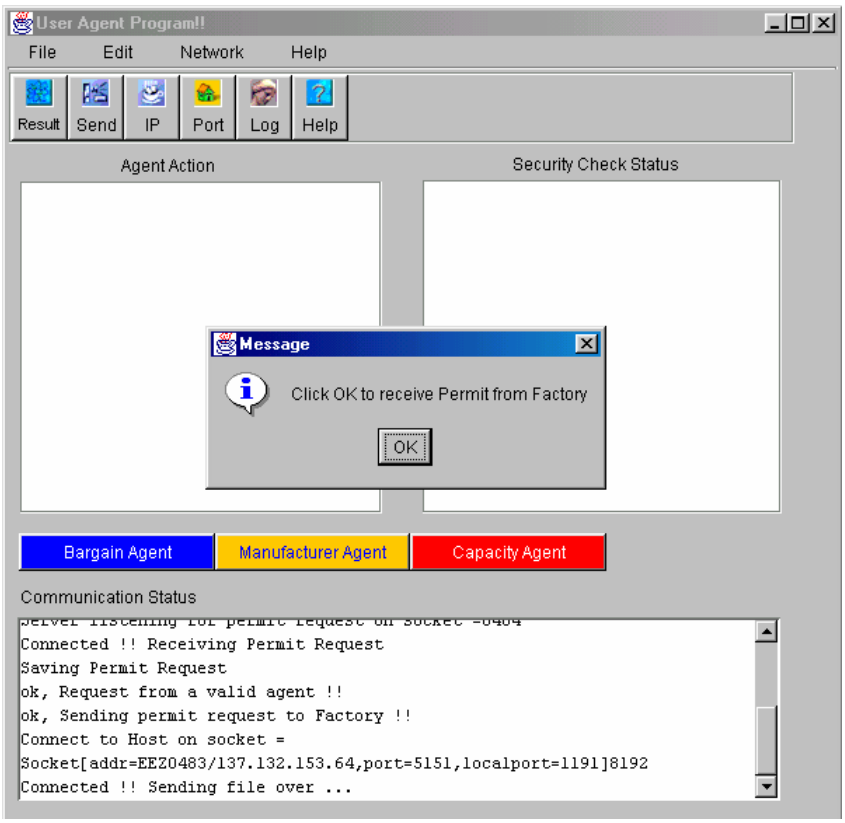

Figure 6. Sample Screen Shot of Agent Butler Receiving a Permit

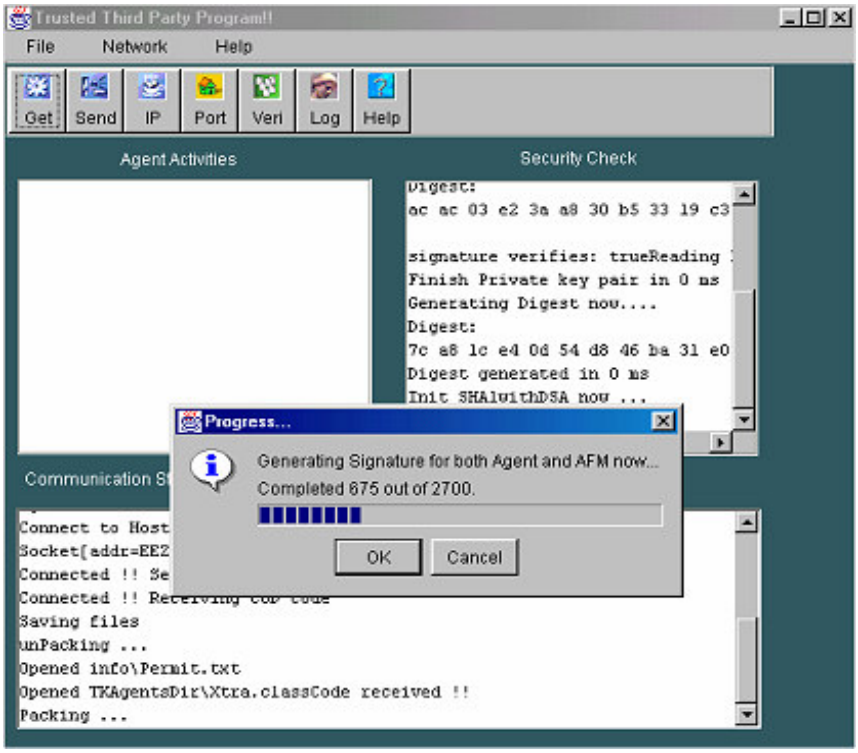

Figure 7. Sample Screen Shot of TTP Signing Agent and Its AFM

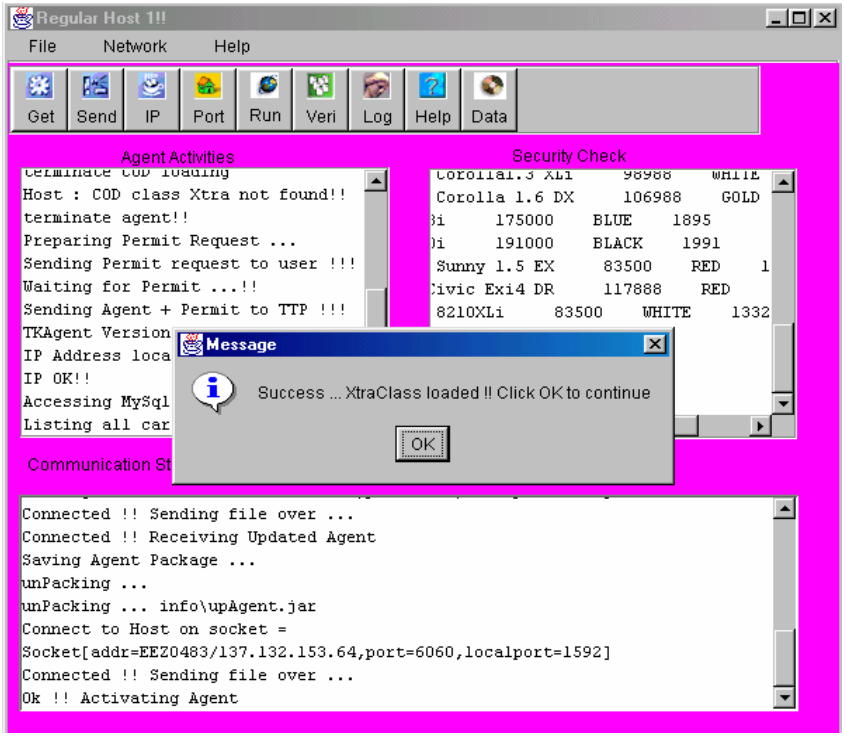

Figure 8. Sample Screen Shot of an AFM Being Loaded Successfully