Secure Electronic Payments Based on Mobile Agents

Artur Romão*
Dep. Matemática - Univ. Évora
artur@dmat.uevora.pt

Miguel Mira da Silva
INESC and IST
mira-da-silva@ip.pt

Alberto Silva
INESC and IST
Alberto.Silva@inesc.pt

Abstract

The SET protocol for secure electronic payments, in particular its purchasing phase, is intended for users connected to the Internet during the entire transaction. This requirement cannot be easily met in high communication costs and/or low bandwidth settings, typically found in mobile computing environments.

In this paper we describe SET/A, a system that works according to the SET rules for purchasing operations without forcing the user to be connected during the entire transaction. This is achieved by sending an agent to the merchant's server carrying all the data necessary to order and pay the goods (products, services or information). The paper shows that this can be achieved safely and efficiently, providing an alternative way for Internet payments using the SET protocol. We give our first impressions on SET/A usage by describing a prototype implementation based on a mobile agent system called AgentSpace, as well as a detailed example of what we consider a realistic application of the system.

1 Introduction

Although the Internet is considered as the privileged environment for electronic commerce, there is still some resistance from the public to buy products and services on-line, and pay for them also on the Internet. For example, by browsing a company's Web server, ordering a product and paying for it filling a form with credit card information. The main reason is that every Internet user has heard of credit card frauds, performed by hackers eavesdropping the connections used to send the data. Even the deployment

*Contact author: Dep. Matemática, Univ. Évora 7000 ÉVORA - PORTUGAL.

of secure servers, based on protocols such as SSL, is not enough, since the credit card information is deposited in the server, where it can easily be read by anyone with access to the server.

The concern for protecting the user's credit card information lead VISA and MasterCard, in association with major software and cryptography companies, into the development of the SET protocol [8]. SET provides important properties like authentication of the participants, non-repudiation, data integrity and confidentiality. Each player knows only what is strictly necessary to play its role. For example, the selling company never knows the buyer's credit card information, and the financial institution authorizing the transaction is not aware of the details of the purchase. Paying for something using a credit card under the SET protocol is more secure than doing it at a restaurant, where the waiter takes the card away from the customer's eyes.

SET is expected to give buyers and sellers the necessary confidence to launch Internet commerce definitively, in spite of some technical and non-technical problems that still exist. That is because SET can be very comfortable from the buyer's point of view, both to use (there will be many SET-compliant software tools to help the users) and to trust (if we assume that financial institutions are able to convince their customers of its security).

On the other hand, SET is a very complex protocol, meaning that it may not be suitable under some technical conditions. In this paper we take a closer look at some computing environments, namely those in which the mobility of users is determinant. Generally, the devices used in these environments have limited computational capacity and use slow, expensive and unreliable connections to the Internet. SET may be too demanding for this kind of equipment and connectivity, rendering on-line transactions unattractive for mobile users. In this paper we propose a lighter mechanism, called SET/A, guided by the SET rules and based on the mobile agent paradigm, to address these problems. With SET/A, the computational burden is taken away from the user's device, and it can be disconnected while the transaction is going on.

We not only describe SET/A and discuss the usage of mobile agents in electronic payments and electronic commerce in general, but we also present our first impressions on an implementation of SET/A, by describing a very simple prototype. This implementations is being built on top of a novel agent system, called AgentSpace, also developed by our group.

The paper is organized in two parts. In the first part, Section 2 describes the SET purchase request transaction and discusses some design issues associated with the usage of SET in a mobile computing environment. In Section 3 we introduce the notion of mobile agents and identify the requirements that are important when using mobile agents for on-line payment systems, and the core of SET/A is described. In Section 4 there is a discussion about some advantages and application scenarios for SET/A. Security issues are

discussed in Section 5.

In the second part, Section 6 presents the AgentSpace system, and Section 7 describes our prototype implementation of SET/A on top of this system. In Section 8 we describe what we consider to be a realistic scenario of SET/A usage. In Section 9 we discuss our views on using mobile agents in electronic commerce in general. Finally, in Section 10 we point out some conclusions and discuss possibilities for future work.

2 Overview of SET

In order to ensure the security of Internet payments, SET provides four basic guarantees to its users [8]:

Data integrity – message digests stored in digital signatures ensure that, for example, the order sent by a customer (the *cardholder*) is the order received by the seller (the *merchant*), preventing accidental or malicious modifications while in transit.

Confidentiality – confidential items such as card numbers and expiration dates are protected so that no unauthorized party (including the merchant) has access to them. Also, a financial institution (typically a bank) authorizing a transaction is not aware of the details of the order.

Authentication – both the cardholder and the merchant are who they claim to be. Furthermore, the merchant can be sure that the cardholder is a legitimate user of a valid payment card; on the other hand, the cardholder can confirm that the merchant has a relationship with a financial institution, allowing it to accept payment cards. These guarantees are accomplished through a public-key infrastructure based on *X.509 certificates* [19] and *certification authorities*.

Non-repudiation – this follows from the authentication guarantee. Before using SET, both the merchant and the cardholder have to register themselves as SET users. Given that their certificates have to be exchanged on each transaction, this prevents them from denying their participation on those transactions.

2.1 SET Purchase Request

The SET protocol is composed of several kinds of transactions, ranging from the certification of participants, to purchase requests, to payment processing. In this paper we are particularly interested in the purchase request phase, which can be outlined as follows (see Figure 1):

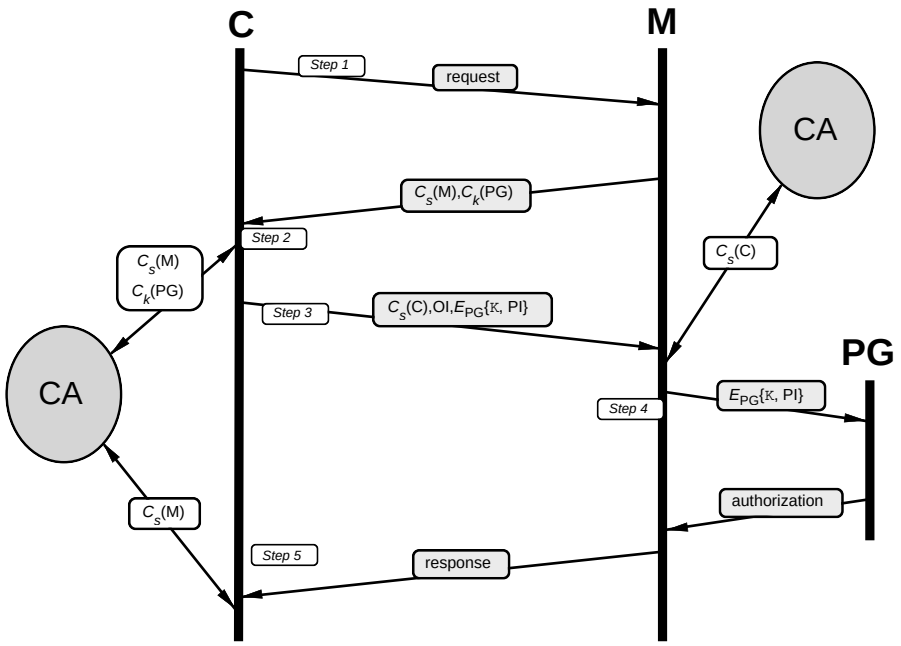

Figure 1: SET Purchase Request Transaction.

Step 1. A *cardholder* (C) looks at a catalog (printed in paper, supplied in a CD-ROM or available on-line on the Web) from a *merchant* (M) and, after deciding to purchase something, sends a request to the merchant's server. The request includes the description of the goods, the terms of the order, and the brand of credit card that will be used for payment.

Step 2. The merchant receives the request and sends back its own signature certificate[1] C_s(M), and the key-exchange certificate C_k(PG) of a *payment gateway* (PG). The payment gateway is a device operated by a financial institution with which the merchant has established an account for processing payments with the brand used by the cardholder. The merchant also sends a unique identifier, assigned to this transaction.

Step 3. The cardholder (i.e., his or her software) verifies the certificates by contacting a *certificate authority* (CA), receives back a confirmation that assures the authenticity and integrity of the data (the merchant has digitally signed it), and creates two pieces of information:

[1]SET uses two distinct asymmetric key pairs, one for *key-exchange* (whose public key is contained in certificate C_k), which is used for encrypting and decrypting operations, and another for *signature* (in certificate C_s), used for verification of digital signatures. Therefore, each SET participant possesses two kinds of certificates, one for each key pair type. A merchant will have a key-exchange and a signature key pair for each card brand it accepts.

- The *order information* (OI), containing control information verified by the merchant to validate the order, card brand and bank identification. The OI also contains a digest of the order description, which includes the amount of the transaction and other elements such as quantity, size and price of the items ordered, shipping and billing addresses, etc. This data, not included in the OI, will be processed outside of the scope of SET.
- The *payment instructions* (PI), containing the amount of the transaction, the card account number and expiration date, instructions for installment payments (if that is the case) and a couple of secret values to prevent guessing and dictionary attacks on the data, among other elements. The PI is encrypted with a randomly generated symmetric key K.

Both elements will contain the transaction identifier so they can later be linked together by the payment gateway. They are further linked by means of a *dual signature*, a mechanism that signs two messages (each one meant for a different receiver) with a single signature operation (see [8] for more details). The signature can later be verified without having to disclose the contents of any message to the recipient of the other.

Then, the encrypted PI ($X_{\mathtt{K}}[\mathrm{PI}]$), and the key K used to encrypt it are encrypted into a digital envelope (E_{PG}), using the payment gateway's public key.[2] Finally, the OI and the digital envelope are sent to the merchant, along with the cardholder's signature certificate $C_s(\mathrm{C})$.

Step 4. The merchant verifies the cardholder certificate and the dual signature on the OI. The request is then processed, which includes forwarding the digital envelope to the payment gateway, for authorization (the details of this operation are outside of the scope of this description).

After processing the order, the merchant generates and signs a purchase response, and sends it to the cardholder along with its own signature certificate.

If the payment was authorized, the merchant will fulfill the order, by delivering the goods bought by the cardholder.

[2] We use the notation $X_{\mathtt{K}}[\mathtt{D}]$ to represent a piece of data D encrypted with key K. The notation $E_P\{\mathtt{K}, \mathtt{D}\}$ represents a digital envelope meant to be sent to some participant P. The digital envelope contains a symmetric key K, encrypted with P's public key-exchange key *k(P)*, and data D, encrypted with K, i.e., the contents of the envelope are $X_{k(P)}[\mathtt{K}]$ and $X_{\mathtt{K}}[\mathtt{D}]$.

Step 5. The cardholder verifies the merchant signature certificate,[3] checks the digital signature on the response, and takes any appropriate actions based on its contents.

The software responsible for the cardholder's side of the protocol manages a data structure called *digital wallet*, where sensitive data like certificates, private keys and payment card information are kept, usually in encrypted files.

The merchant will have a more complex system composed of several parts, doing different jobs: managing the dialog with cardholders, signing messages and verifying signatures and certificates with CAs, asking payment gateways for payment authorizations, and so on.

2.2 Difficulties with Mobile Computing

Wireless networks, especially cellular phone networks, have known explosive growth over the last few years, reflecting a world in which it is important to be able to communicate regardless of the location. As the Internet becomes more and more important for business transactions, it is natural that wireless technology is used to connect to this global network. The possibility of having all the resources and benefits offered on the Internet available while being away from home or office is particularly attractive. In mobile computing it is desirable to have all the facilities usually found on the Internet, namely the possibility of acquiring and paying for products and services.

The kind of mobility we are interested in is based on portable devices, which have limited computing capacity and/or limited connectivity. For example, computing-capable mobile phones, PDAs, hand-held PCs, up to notebooks connected to the Internet through GSM modems.

2.2.1 Bandwidth and Cost

There are several issues regarding the conditions in which mobile computing takes place. In the context of this paper, we are mainly concerned with the following:

Low bandwidth – terrestrial wireless network protocols such as GSM or satellite-based systems like IRIDIUM [10], typically offer bandwidths in the range of 2,400bps to 9,600bps (although there are claims for much larger bandwidths with broadband satellite systems in the future).

Poor connectivity – the connectivity based on these systems is generally of low quality, with high error rates. These factors make it difficult to handle long, connected sessions, transferring large amounts of data.

[3] This second verification is necessary because in the mean time the certificate may have expired or been revoked for some reason.

High cost – using a cellular phone or a satellite-based connection is generally more expensive than through a traditional analog connection or ISDN. On the other hand, poor connectivity raises costs, since it leads to longer on-line sessions.

In mobile computing, bandwidth capacity is thus proportionally inverse to the cost, and this fact has to be taken into account when designing applications for these environments. *It is the combination of the limitations described above that makes the rationale for the rest of the paper.* Although bandwidth is important, it is not the only, and maybe not the most important factor—as stated above, this kind of problem tends to be solved in the future.

2.2.2 SET in a Mobile Environment

In an error-prone environment such as GSM or any other used for mobile communications, a user shopping on the Internet and trying to pay using SET-compliant software may experience several connectivity problems during the payment operation. It is easy to imagine how frustrating it can be for the cardholder to deal with a series of connection interruptions, even if there are recovery mechanisms. This is further complicated by the accumulation of state information both in the wallet and in the merchant's server, in order to let the transaction proceed. Even if it eventually succeeds, the overall cost has probably been too high.

What is needed is some kind of asynchronous mode of operation, in which the cardholder can send the request, disconnect, and re-connect later to receive the response from the merchant. This reasoning seems to suggest a typical message-passing mechanism, but this is not suitable, for two reasons:

1. Three of the five steps of the purchase request transaction described above are executed on the cardholder's side. As such, there would have to be *two messages* (one to send the request, and another to send the OI and the digital envelope) after receiving the first response from the merchant. Since the cardholder can be disconnected for an arbitrary long period after sending the first request, the whole transaction would have to wait for this intermediate synchronization to happen.

2. On the other hand, if there would be a way to avoid this step and send a single message, this would mean disclosing sensitive information (e.g., credit card number) and letting it be used by a remote server in an unpredictable way. This is because the cardholder would not know the merchant's and payment gateway's key-exchange keys, so the information could not be encrypted.

This means that it is necessary to reduce the cardholder's role to two steps, the initial request and the final receipt of the response. The request

sends all the necessary information to complete the transaction successfully, but in such a way that the remote system can never take control of it.

We conclude that, in environments like these, the SET protocol may benefit from an alternative agent-based implementation. The cardholder may send an entity (an agent) with enough information for processing the entire transaction and, at the same time, capable of hiding the sensitive data from the outside.

3 SET/A: An Agent-Based Payment System

In this section we introduce the concept of mobile agent and propose a secure Internet payment system based on this model and on the SET protocol. We concentrate on the purchase request transaction, as this is the most important phase (from the cardholder's point of view) and the only one that really makes sense for our system. The other SET transactions refer to situations in which there isn't the notion of a buyer paying a seller.

3.1 Mobile Agent Applications

A mobile agent can be defined as a software element (program, procedure, object, thread) owned by a user or another software element, capable of migrating between machines in order to execute a set of tasks on behalf of its owner.

Mobile agents are said to be autonomous, in the sense that they can make their own decisions while away from their host. This implies that a mobile agent (agent, for short) is not just a piece of data (i.e., a message) being transferred between systems, but may also carry some logic (i.e., code) and execution state. This enables the agent to perform parts of its tasks in one system, migrate to another and continue its work there.

Agent technology has been receiving growing interest from the research community and has matured significantly in the last few years [14, 20]. However, the number of applications using this technology is still scarce [6]. Electronic commerce is generally seen as one of the most promising application fields for mobile agents [2, 1].

For example, the literature typically refers to a buying agent, leaving its host with the mission of querying several vendors about a certain product, determine which one offers the lowest price (or some other kind of preferential condition) and buying from this one, paying for the product. Clearly, there is the perception that agents are suitable for this kind of activity, and the *ability to pay* is one of the desired properties they should have. The major concern is always how to do this in a secure way, in particular without revealing confidential information to the outside.

In this paper we describe a payment system called SET/A, based on the SET protocol. SET/A is implemented using an agent traveling from the

cardholder's computer, carrying all the relevant information, to an external server (e.g., the merchant's). On arrival, the agent assumes the cardholder's role and carries on a complete purchase request transaction with the merchant. From the merchant's point of view there will be no distinction between an agent and a real (i.e., human) buyer; a SET purchase request is being performed with another entity, which represents a cardholder with a valid certificate and payment data.

3.2 Agent Requirements

In order to be able to do its job, there are a few requirements that the SET/A agent must fulfill:

- **Small-sized** – since we are assuming low and/or expensive bandwidth, we have to minimize the time it takes to send the agent. On the other hand, transport media with low reliability makes it difficult to transmit long streams of data. For example, a reliable transport protocol like TCP would require many re-transmissions [7]. Clearly, the agent should be as small as possible for better performance and smaller costs. This can be a problem if one needs sophisticated agents, as will be discussed below.

- **Survive inside hostile execution environments** – the cardholder wants to be sure that the agent will be able to complete the transaction as expected, without being disturbed by any external factor. This requires a relative degree of tolerance to faults on the external server, where the agent will execute, as well as immunity to attacks trying to force the agent to make unwanted decisions or perform unwanted actions.

- **Hide confidential information** - one of the main purposes of SET is to offer an appropriate level of security so that the cardholder can be sure that none of the confidential data is disclosed to any unauthorized party. Thus, SET/A has to ensure that the confidential data the agent carries is kept private.

3.3 Purchase Request

SET/A is meant to implement the purchase request phase of SET using mobile agents. It should be noted that SET/A is designed to be as compatible with SET as possible, only requiring significant modifications on the cardholder's side. The merchant's software remains unchanged, since its interaction with the agent is the same as it would be with the cardholder.

A purchase request under SET/A has a few more steps than in SET, since the cardholder has to send an agent, and the agent has to come back

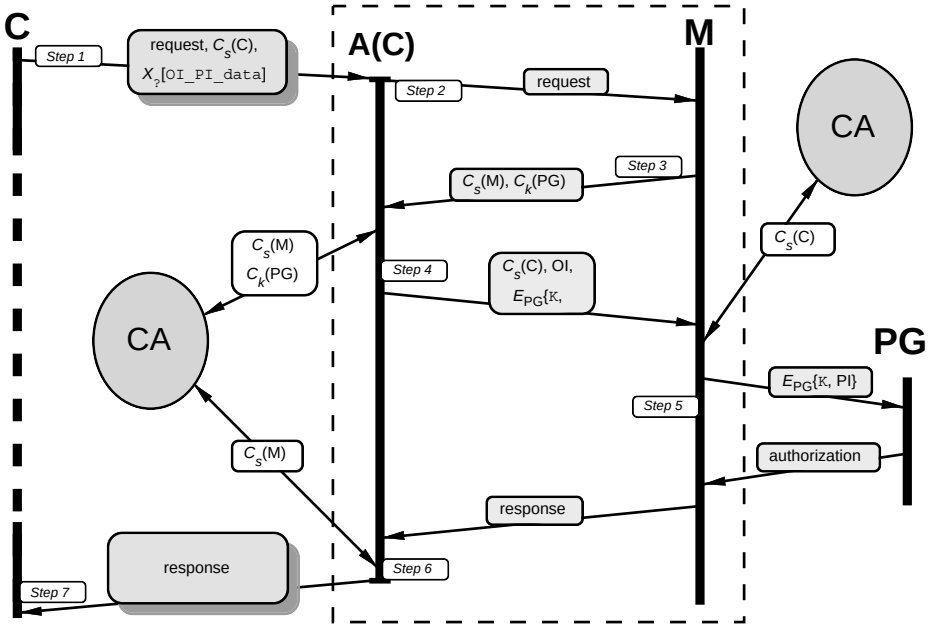

Figure 2: SET/A Purchase Request Transaction.

to the cardholder's computer when the transaction is done. SET/A purchase request is described below (see Figure 2):

Step 1. As before, the cardholder C chooses a merchant and builds a request with the same elements as in the original SET request. Then, an agent, A(C), is sent to an external server,[4] carrying the request, the cardholder's signature certificate C_s(C), the account information and other data needed to compose an OI and a PI (we call it `OI_PI_data`). Clearly, this data has to be protected somehow, but we will deal with that later (see Section 5). For now, we will just assume it is protected by encryption with someone's key ($X_?$[`OI_PI_data`]).

Step 2. After arriving at the host server, the agent sends the request to the merchant M (i.e., its order processing software).

Step 3. The merchant returns a signed message with its signature certificate C_s(M), the payment gateway's key-exchange certificate C_k(PG), and the transaction identifier to the agent.

Step 4. The agent verifies the certificates and the signature, creates the OI and the PI and generates a dual signature on them. Next, it generates

[4]This may be the merchant's server, or any other server, as will be discussed below. For simplicity, we will called it *host server*.

a random symmetric key K and uses it to encrypt PI. Finally, the payment gateway's public key is used to create the digital envelope E_{PG}, containing the encrypted PI ($X_{\mathtt{K}}$[PI]), and the key K. (The security considerations regarding these data are similar to those discussed in Section 5, so we will ignore it for now.) This digital envelope, the OI and the cardholder's certificate C_s(C) are the sent to the merchant.

Step 5. The merchant verifies the certificate and the dual signature on the OI, and then proceeds as described above in step 4 of the SET purchase request (see Section 2.1), sending a signed response along with its own signature certificate.

Step 6. The agent receives the response, verifies the certificate and the signature, and migrates back to the cardholder's computer.

Step 7. The agent arrives at the cardholder's computer carrying the response from the merchant (see Section 7.4 for alternatives for this rendezvous to take place). The cardholder's software then proceeds as in SET's final step.

4 SET/A Discussion

We designed SET/A with mobile computing in mind, especially focused on the two factors that we have been emphasizing as determinant: low bandwidth and expensive connectivity. Our proposal of adopting an asynchronous computing model is a natural way to overcome those limitations.

Below we discuss the advantages and identify possible scenarios in which the usage of a payment protocol designed for disconnected settings is the best way, or the only one that makes sense economically, to do shopping on the Internet in a mobile computing environment.

4.1 Processing Capacity

PDAs and mobile phones have limited processing capacity, and can hardly handle processor-demanding activities, such as those involving cryptography (key generation, encryption, signature generation and verification). Therefore, if SET is to be used on this kind of devices, either the CPU capacity has to be increased, or the load has to be transferred to an external machine.

SET/A solves this problem by performing all the cryptographic work outside the cardholder's device, on a machine that is supposed to be able to handle many of these activities concurrently.

4.2 Minimizing Costs

One of the problems of electronic payments is the relative high cost of small-value transactions. Using a mobile device to access the Internet and perform a small-value purchase, setting up the connection and maintaining it opened for as long as a SET-based transaction takes place, may have a cost similar (if not higher) to the value of the transaction itself. Having to pay the double of the price of a product, however low it may be, is enough to discourage consumers to use this kind of connectivity for their shopping.

Using SET/A for this kind of payments, the connectivity costs are limited to the time the agent is sent, and when it has to be received.

4.3 Applications

Despite the advantages of SET/A, it can be difficult to imagine a complex purchase transaction requiring a lot of interaction with the cardholder (during the phase in which the OI and PI are composed), being handled by SET/A. This can be the case when the merchant offers a set of alternatives concerning payment or delivery of goods, from which the cardholder will have to choose. Letting the agent decide would be a possibility, but that would require more sophistication, i.e., more code, hence a bigger size. Nevertheless, it still would not guarantee that the agent would be prepared to make the appropriate decision in every situation.

We consider SET/A more appropriate when the contents of the OI and PI are predictable, even if the agent is not carrying them with it when migrating to the external host (i.e., it has the ability to compose them later). The simpler scenario is that in which the elements needed to complete the transaction are well defined and can be included in the data the agent is carrying while going away from the cardholder's device. In other words, the buyer has an approximate idea of what to buy and how much it will cost.

Of course, a moderated degree of sophistication can be acceptable in terms of agent size, and it may be used in situations where the possibilities are known in advance. For example, the agent could have enough knowledge to decide what should be the delivery method, depending on the amount of money it is allowed to spend, or other simple algorithms.

This reasoning does not imply that the agent should simply act as a proxy for the client. The fact that the agent is capable of making its own decisions (even if very simple), makes it a more powerful mechanism than a simple proxy. In Section 8 we provide an example that illustrates this claim.

5 Security Issues

One of the obvious concerns in a network payment system is to protect the user's critical data, in particular the credit card information. SET's usage

of the dual signature mechanism and the encryption of the PI and account information (into a digital envelope with the payment gateway's public key-exchange key), ensure the necessary privacy of critical data. In particular the data is protected from a potentially hostile environment, such as the computer system in which it is going to be used. In the case of SET/A, the server where the agent will be executing may be considered a hostile environment, in the sense that any malicious action from that host against the agent may compromise its secret data.

Of course, protecting the host from the agent is also important, but that concern is out of the scope of SET/A. Our system is intended to provide a service to the cardholder, through an agent. For that, we have to assume that the agent behaves as expected in order to execute the protocol described in Section 3.3, otherwise it will fail to play its role properly. Therefore, the agent will not perform any unwanted action that may compromise its host server and/or the merchant. Thus, the protection of hosts against malicious agents is not an issue in this paper. Anyway, SET/A may incorporate any of the mechanisms that already exists, in order to help providing this kind of protection (see [13, 11, 5], for example).

5.1 Protecting the Agent

For SET/A to be able to ensure the same level of protection as SET, without modifying the description outlined above, it must be possible for the agent to carry and use classified information without having to disclose it to the wrong entities. Also, the generation of the symmetric key `K` has to be performed in such a way that no one other than the payment gateway has knowledge of it.

Protecting the agent from hostile environments is a major research issue in mobile agent security, and we may consider applying some of the existing proposals to SET/A. For example, running the agent in a *tamper-proof environment* [23] or a *secure coprocessor* [24] seem promising possibilities. The agent would migrate to a protected (hardware) environment, securely attached to the external server (e.g., the merchant's), and all the confidential data would be handled inside this environment. This solution would increase the security level at the cost of an additional investment in hardware. In Section 8 we give an example that illustrates the usage of SET/A in an environment protected this way.

On the other hand, replay attacks making the agent pay more than once have to be prevented. A possible approach is suggested in [22], in which hardware protection is combined with mechanisms ensuring a limited agent lifetime and storage of the agent's identification in protected, non-volatile memory.

The "software alternative," in which the agent executes inside the external server without any hardware protection, requires some kind of wrapping

to hide the secret data. Cryptography is the natural wrapper, and some initial steps to allow the execution of *hidden computations* are being taken [15].

5.2 Protecting the Agent's Data

If we relax the requirement of following closely the SET protocol for purchase requests, there may be a better way to achieve the goal of protecting the data. First, recall that the data we want to protect is to be encrypted with the payment gateway's key. Suppose that the cardholder knows in advance which payment gateway the merchant is using for the card brand, and the OI and the PI can be built in advance. Then the process of generating the dual signature, the random symmetric key, and the digital envelope can be performed before the agent leaves the cardholder's computer. When it migrates, all the information, completely protected, can now go with it, and the agent only has to give it to the merchant and wait for the response.

This approach has two major drawbacks: first, as we mentioned earlier (see 4.1) the cardholder will typically use devices with limited computing capacity. Thinking about performing all the cryptographic operations mentioned in the previous paragraph may be unrealistic.

On the other hand, to obtain the payment gateway's certificate in advance, the cardholder has to perform an initial request to the merchant, wait for the response, and then proceed with the agent. This is similar to the message-passing protocol described in section 2.2.2, and has the same disadvantages. Furthermore, the agent's role would be limited to that of a "messenger," delivering the request, and bringing back the response, which is, again, no more than message-passing. (Anyway, the core of the transaction would still benefit from being performed by the agent, which would be, in principle, closer to the merchant's server.)

Note that the merchant software could still remain unchanged. In step 4 of SET/A the agent would receive both certificates, but only would have to verify the merchant's. In our opinion, this small difference in the agent's behavior and the cardholder's initial request to obtain the payment gateway's certificate, are two minor changes to the cardholder side of the transaction, and perfectly acceptable given the improved security they offer.

Both approaches described in the previous and in this section need to be further investigated, in order to find a good compromise between them. The first one, based on real mobile and autonomous agents, is more challenging in that it opens a wider spectrum of possibilities. Electronic commerce is a promising application field for mobile agents, and many research initiatives are undergoing. We expect to benefit from those efforts in order to improve the model and solve the difficulties described above.

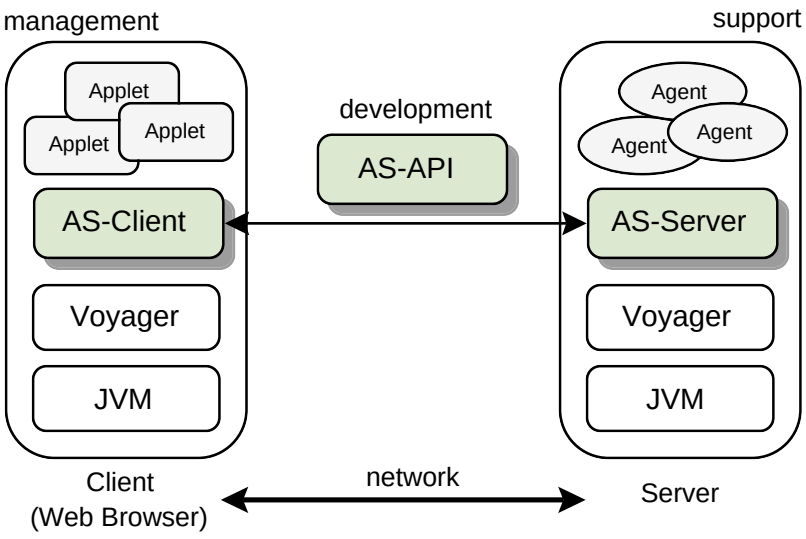

Figure 3: AgentSpace main components.

6 AgentSpace Overview

In order to validate our ideas, we have built a SET/A prototype implementation. This prototype is based on a working mobile agent system that we have also designed and implemented (in the scope of another project), called AgentSpace [16, 17, 18]. This novel agent framework is based on ObjectSpace's Voyager [12], a Java-based CORBA system for supporting distributed objects. In this section we will present a brief overview of AgentSpace. A more in-depth description regarding the system's implementation and API can be found in [18].

6.1 Architecture

AgentSpace's main goals are the support, development and management of agent-based applications. These goals are provided through three separated but well-integrated components, as depicted in Figure 3. Both server and client components run on top of Voyager and Java Virtual Machine (JVM), and they can execute in the same or in different machines. An agent always runs on some AgentSpace server (AS-Server) context. On the other hand, it interacts with its end-user through (specific or generic) applets running in some Web browser's context.

The AS-Server is a Java multi-threaded process in which agents can be executed. An AS-Server provides several services, namely: agent and place creation; agent execution; access control; agent persistency; agent mobility; generation of unique identifiers; support for agent communication;

and, optionally, a simple interface to manage/monitor itself.

The AgentSpace client (AS-Client) supports, depending on the corresponding user access level, the management and monitoring of agents and related resources. The AS-Client is a set of Java applets stored on an AS-Server's machine in order to provide and adequate integration with the Web, offering an Internet user the possibility to easily manage his own agents remotely. Furthermore, the AS-Client should be able to access several AS-Servers, providing a convenient trade-off between integration and independence between these two components.

The AgentSpace application programming interface (AS-API) is a package of Java interfaces and classes that define rules to build agents. In particular, the AS-API supports the programmer when building: (1) agent classes and their instances (agents) that are created and stored in the AS-Server's database for later use; and (2) client applets that provide the end-user interface to remote agents.

These clients/applets may be either generic mini-applications—such as the AS-Client itself—or specific to some particular agent.

6.2 Main Object Model

AgentSpace involves the support, development and management of several related objects: context, place, agent, user, group of users, permission, access control lists (acl), security managers, tickets, messages, and identities. Figure 4 shows the relationships between these objects through an UML class diagram [4].

The context is the most important and critical object of the AS-Server, as each AS-Server is represented by one context. Specifically, the context contains the major data structures and code to support the AS-Server, such as lists of places, users, groups of users, meta-agent classes and access control lists.

Each context has a number of places. The execution place, or simply place, has mainly two objectives: First, to provide a conceptual and programming metaphor where agents are executed and meet other agents. Second, to provide a consistent way to define and control access levels, and to control computational resources.

The place has a unique global identity and knows the identification of its owner/manager. Optionally, places can be hierarchically organized. The place can also contain the current and the maximum number of agents allowed, in order to support some resource management. In order to keep track of its agents, the place maintains a list containing its native agents and another with its visiting agents. The place also knows in which place its native agents are executing at a given point in time.

The agent is the basic element of the system. Agents are identified by a unique and global identity. Agents are active objects that execute in some

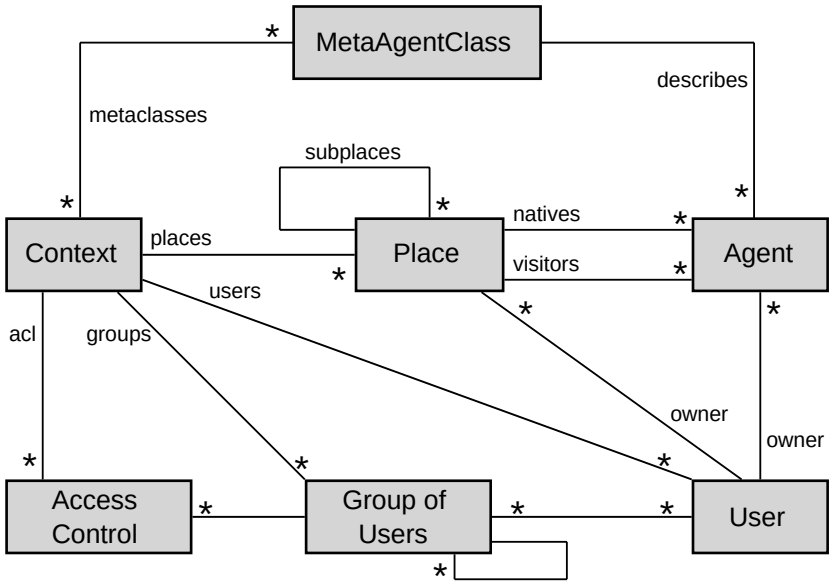

Figure 4: Class diagram of AgentSpace main concepts.

AS-Sever, but from a conceptual perspective, they are currently in some place. Agents may navigate to other (local or remote) place if they have permission to do so.

Only one user owns an agent. Nevertheless, other users (or even agents from other users) might interact with it, if this is granted by the agent's security policy.

The AS-Server also maintains lists of users, groups of users and access control lists to implement the permission and access control mechanism.

7 Implementing SET/A on AgentSpace

Our first implementation of SET/A uses AgentSpace for agent mobility. This choice has two goals: to have a working prototype of SET/A, in order to validate the model against a practical environment, and to provide some feedback on the applicability of AgentSpace for this kind of usage. The resulting prototype is very simple and limited in functionalities, and there are no good figures for items such as performance or agent size, yet.

In this description it is important to distinguish between the human-agent interface (via a Web browser), and the communication amongst agents via remote calls (typed objects) or message passing (strings). Our prototype deals with both issues, but focuses on the later. The former is important because it is necessary to ask the user for the data the agent needs (credit card details, amount of money, etc.).

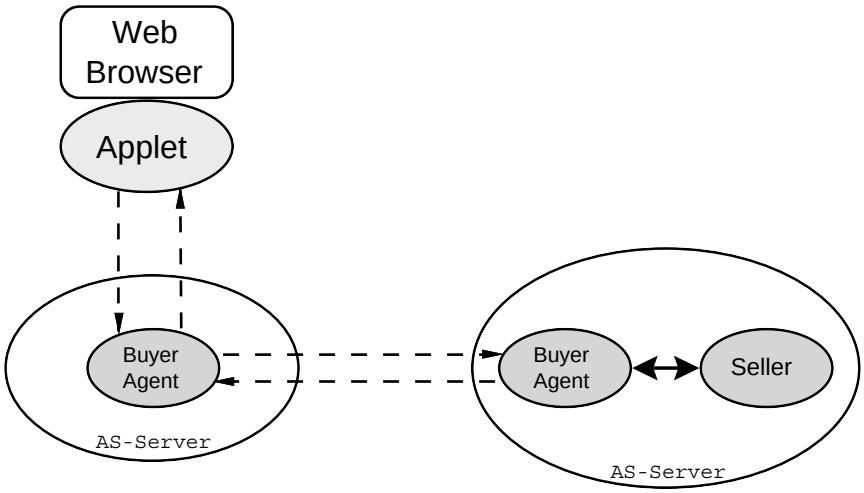

Figure 5: SET/A on AgentSpace architecture.

7.1 Prototype Architecture

In order to use SET/A for payments, the cardholder will need some kind of interface to interact with the agent. This interface should be an applet (running on a Web browser) or another agent. In our experiment, the user interacts with an applet, filling in the credit card details using an HTML form and clicking a "Go & Pay" button to transmit the data to the agent.

The applet implementing this interface uses the AgentSpace framework to control the (local or remote) agent, being responsible for initializing the agent and context views, and other "administrative" data. When the relevant data is ready, the agent is created, then it receives the data from the applet, migrates (in our case, to the merchant's machine), performs the transaction by interacting with the merchant's SET software and returns to the cardholder's machine. Figure 5 illustrates the architecture and the following sub-sections elaborate on the agent's activities.

7.2 Sending the Agent

The agent packs the data (i.e., assigns the appropriate values to its variables) received from the applet, obtains the identifier of the remote (the merchant's) place, acquires a ticket to be accepted there an issues a `moveTo()` operation to migrate to that place. The name of the method to be invoked upon arrival at the merchant's place is provided as an argument to `moveTo()`. Migration can occur immediately or be delayed until the machine is connected to the Internet. (We are also investigating asynchronous forms of migration with MIME-enabled mail messages.) This process is illustrated by the Java pseudo-code in the Appendix.

7.3 Performing the Transaction

The agent arrives at the merchant machine to perform a SET transaction. The method indicated in the call to `moveTo()` is then invoked and the transaction proceeds. First, the agent sends the request to the merchant and waits for the response (this is simply a confirmation that the transaction may continue, since we don't support the certificate operations yet).

The transaction may or may not involve prior negotiation (for prices, guarantees, etc.) and delivery (in the case of digital goods). If so, the agent is expected to bring all the data needed to carry on the negotiation. Another possibility is letting the agent have some party at the cardholder's place with which it can communicate (securely) and ask for some elements it needs to negotiate with the merchant Since this contradicts the motivation for using the agent and disconnect from the Internet, we didn't explore this possibility any further.

At this stage we chose to ignore negotiation, given its complexity—we have to devise or use some existing mechanism to structure the various conversations (probably based on languages such as KQML [9]), establish secure channels, etc. Anyway, negotiation and delivery is outside the scope of SET and, thus, of SET/A.

The agent builds and sends the OI and the PI to the merchant and waits for an authorization (always granted, in the current implementation). After that, the payment transaction is done.

7.4 Agent Return

When the agent has paid, it will migrate back to the cardholder's machine. However, the machine may be disconnected from the Internet (it may be a hand-held PC or a mobile phone). This is not a problem, since AgentSpace has a persistence mechanism [18], which guarantees that the AS-Server keeps the agent's state as long as needed, even in the case of failures.

In our prototype, the agent attempts to migrate to the cardholder's computer as soon as it is ready. There is no exception mechanism to cope with network failures However, in a real implementation we anticipate at least four other scenarios:

1. The agent polls the cardholder's machine (i.e., "home") from time to time and decides to migrate as soon as it gets a connection. This requires that the agent maintains a reference to its place of origin, but this is how many protocols work and should be acceptable up to a certain point.

2. The agent quietly waits for a call from its owner to come back. When the cardholder re-connects to the Internet, all agents native to this place that are away are contacted.

Agents that have finished their work will return, others just respond "I'm still working," and probably still others will not respond at all, either because they have moved, they have died, or any other reason. In the case an agent has moved, it may use another method (among those described here) to return.

3. The agent migrates to a well-known, highly available intermediary—such as an Internet service provider—where it waits to be called by its owner. An AS-Server might have a buffer of agents ready to migrate, waiting for activation from their owners' places/AS-Servers.

   This is a variant of the previous scenario, and the owner only needs to call the intermediary and receive the agents that are waiting there. This is very much like the delivery mechanism of SMS messages to cellular phones. Of course, this raises several issues related to security and cost, that need to be further investigated. See section 8 for an example, where this issues are explored in detail.

4. The agent packs itself as a MIME mail message and sends itself to the cardholder machine. Using this approach, the owner will receive the agent back as an attachment to a mail message, that should be opened either to be read, or to continue executing. This scenario does not use AgentSpace for coming back. Nevertheless, the agent enjoys all the advantages of Internet mail, in particular asynchronous delivery. We are currently working on this model of migration.

After returning, it is convenient that the agent tells its owner what happened while away. This is a user-interface problem similar to telling the agent what to do, although this time it is the agent that takes the initiative.

In our experiment, the agent just sends a message to the applet executing in the Web browser. However, we expect that in many cases the owner will not be waiting for an answer, so it is necessary some other form of communication. Here are some alternatives:

- The owner may check a Web page representing the agents from time to time (very much as mail messages have to be checked).
- Another alternative is to instruct agents to send their results as mail messages, since cardholders probably check their e-mail regularly.
- In a third alternative, the agents will send messages using other communication media, such as the cardholder's pager or mobile phone.

8 Usage of SET/A

We now present an example in order to illustrate the usage of SET/A in a realistic scenario, and to clarify what are the applications and computing

environments we consider appropriate for SET/A.

Until now, we have been careful in not forcing the agent to migrate to the merchant's server, rather it may communicate with the merchant from any other external server.[5] This avoids having to put a higher degree of trust in the merchant, which could be a limitation to the confidence of the cardholder in order to use SET/A, and also a limitation to the security of the agent. We prefer to use the notion of a *trusted third-party*, so that the cardholder may be confident that his agent will not be compromised. Additionally, we propose the usage of hardware protection (see Section 5.1) in order to increase the security level.

In this example, the cardholder has a mobile phone equipped with a processor that enables it to hold data and code (for convenience, we will assume this is Java code), that will be used to build an agent. This processor is also able to run a subset of AgentSpace that enables it to send and receive agents. This subset runs on a limited JVM (such as Waba [21]) and the AgentSpace capabilities are reduced to the minimum.

The agent is sent to a server operated by the cellular phone provider, and all the interaction with the merchant will be conducted from this server.

In order to minimize the size of the agent when it leaves the cardholder's mobile phone, it only carries two types of elements:

Secret data, such as the card account number and expiration date, and the cardholder's private signature key.

Specific code, in the form of Java classes implementing whatever logic the cardholder wants his agent to have. This code, together with some additional data (e.g., auxiliary variables), may be used by the agent, for example, to negotiate the deal with the merchant.

The cryptographic and the SET-specific classes are kept at the cellular phone provider's server, inside secure hardware. For maximum security (i.e., trustworthiness), these classes are kept in *jar* files, signed by their developers. The secure processor also has two, well-known certificates, one for signing and another for encrypting—in fact, it could be just one certificate, but we will adhere to the SET's policy of having different certificates for different operations.

Before leaving its home environment, the secret data is encrypted with the secure processor public encryption key. Optionally, the agent's specific code may also be encrypted, thus providing maximum privacy. This may not be a wise choice, given the limited processor capacity available in the mobile phone. Nevertheless, encryption of data, and possibly code, when sending the agent is a price that the cardholder's device will always have to pay.

[5] Of course, the agent could migrate into the merchant's server, but this is only a particular case.

Upon arrival at the cellular provider's server, the agent installs itself inside the secure environment. It then loads the cryptographic and SET classes, and verifies their signatures if needed. The agent then provides the secret data it carries (and maybe the code, as well), for decryption by the secure processor. Thus, a package is built with the complete set of functionalities a SET cardholder wallet is expected to have.

From this point on, the agent is able to conduct a SET purchase request transaction, with the privacy guarantees it needs to have in order to protect its owner's secret data. The specific code enables the agent to implement its owner's policies when talking to the merchant (e.g., for negotiation).

When the transaction completes, the merchant's response may optionally be encrypted before the agent returns to its owner (in this case, the owner's key-exchange public key must be available). This response, and a few classes that may be needed by the agent for returning home, are the only elements that will be sent from the cellular provider's server, minimizing the agent's size on its return.

The return itself happens when the cardholder re-connects to the Internet and contacts the provider, so that its agents return home. This is a variant of scenario 3, in Section 7.4.

This example proves that if the agent is provided a reasonably secure and trusted environment, it is perfectly capable of performing the whole transaction securely. The server is operated by the cellular phone provider mostly for technical convenience, but it could also be operated by some other, more trusted third-party, such as a notary.

The fact that the generic classes (SET protocol and cryptography, i.e., the "heaviest" ones) are not transported by the agent, contributes to minimize its size. On the other hand, the cardholder's own requirements can be implemented by the agent's specific code, thus providing a high degree of flexibility.

It is worth emphasizing this last issue: If all purchase transactions were to be performed in the same way, then there would be no need to use mobile agents. The same results could be achieved by a proxy for the cardholder on the remote side. It is the possibility of the cardholder to satisfy his very own requirements (most notably, his algorithms for negotiation) that require the transfer and remote execution of code on his behalf. In other words, *a mobile agent-based solution.*

9 Mobile Agents and Electronic Commerce

Secure on-line payments is just one of the issues in electronic commerce in which the mobile agent paradigm may help. In this section we present our views on this subject.

Apart from paying on-line with mobile agents, the techniques explored

in this paper can be used in many other applications. Here we name just a few areas in which we believe agents can be used to look for, select, deliver, pay, and protect data and services.

- Selling information on the Web – decrees, laws, internal memos, experts' reports, and so on.
- Agent-based information brokers – newspapers, magazines, document databases in general.
- Electronic contract negotiation – agents carrying contracts and contract editors [3, 6].
- Electronic data interchange (EDI) – the data flow observed in EDI transaction is similar to that of SET (see below).

In our research we intend to explore EDI a bit further since it has enormous potential not only to be enhanced with electronic payment, but especially for electronic commerce in general.

EDI messages include orders, invoices, deliveries, and other business messages, as well as various acknowledgements for these messages. A typical individual transaction, as seen from the seller's point of view, follows this pattern:

1. *order* – the seller receives an order from a buyer, checks, processes and stores it, and finally sends an acknowledgement to the buyer with a delivery date of the good.
2. *invoice* – the seller receives a confirmation that the good was delivered, stores it, and sends an invoice to the buyer and stores a copy locally.
3. *payment* – the seller receives the payment (usually outside EDI).

From the buyer's point of view there is an equivalent sequence of receive/send messages, all about the same transaction. It would be very convenient if the entire transaction could be modeled as a single mobile agent traveling back and forth between buyer and seller.

With SET/A, the payment phase—not currently part of EDI—could be integrated with the rest of the transaction as well.

10 Conclusion

SET is expected to gain wide acceptance as a secure Internet payment system, since it combines the well-known credit card payment method with an elaborated security protocol. SET is aimed at providing the necessary security through the authentication of the participants in a commercial transaction, as well as privacy of financial information. The fact that SET was

developed and is being pushed by the major credit card companies is yet another factor contributing to its wide acceptance.

However, SET is a very complex and "heavy" protocol, and if from the cardholder's point of view it may be generally simple to understand and use, its complexity may prove it unsuitable for some computational environments.

In this paper we discussed the usage of SET in mobile computing settings and argued that it is not practical to be used in this kind of environments. The low bandwidth and high connection costs generally associated with mobile computing make it necessary to devise mechanisms that adapt better to those conditions, without loosing any of the benefits SET offers, especially to cardholders.

Based on these requirements, we proposed SET/A, a payment system based on the SET protocol and the mobile agent model. The purchase transaction, the most important part of SET from the cardholder's point of view, is implemented in SET/A almost as in SET. The cardholder's role is now played by an agent, which is sent to the outside, "closer" to the merchant's server, with the relevant information to perform the necessary operations. Since the cardholder doesn't need to keep the connection opened while the transaction is running, this solution contributes to lower cost and higher robustness.

We also discussed the need for small-size agents due to limitations on the bandwidth of the cardholder's device, and the possibility of providing the agent with enough information so that its role is not limited to hand the data over to the merchant and collect the response. The requirement for mobility makes us pay special attention to the size of the agents we want to produce, in order to find a suitable compromise between the amount of data and code they need to do their work and the limitations imposed by the media through which the agents travel. However, we are also interested in keeping the agent as intelligent and autonomous as possible, allowing it to make its own decisions (even if very simple) when necessary.

Security is another important issue, as the agent will need to take confidential information with it. The autonomy requirement implies that the agent must be able to have access to this data and act according to its contents, not just being a passive carrier. Even if the information is encrypted, the agent will need to be able to decrypt and use it, yet hiding it from others, which is very difficult when these actions are performed inside extraneous, potentially malicious servers. Thus, we are particularly interested in future advances in this field that can be incorporated into SET/A.

Many people have suggested that mobile agents are a solution looking for a problem, yet many others have suggested that electronic commerce is a problem looking for a solution. In this paper we have shown that the idea of using mobile agents for electronic payments makes sense. Having agents paying for goods and services using SET provided an interesting problem and

respective solution. Furthermore, Section 9 suggested that the same kind of agent-based solution can be generalized to many other similar problems.

Our experiment with AgentSpace, although very simple and limited in scope, works well and gave us a very first perspective of how effective mobile agents can be in this kind of applications. This implementation was developed in the context of electronic commerce, more specifically for information brokers and contract negotiation in the scope of the COSMOS project [3]. As a side result, we expect the work with SET/A will improve our agent system so that it can be used in a variety of environments.

In the future, we will develop further the AgentSpace system, investigate the concept of MIME-based mobile agents, and experiment with other application areas such as EDI and electronic contract negotiation. We will also continue our research in SET/A. Presently, we are building a SET cardholder implementation, to enable our agents to perform real payments.

Appendix

The code below shows an agent-based applet which presents an interface where its users enter the remote place where some agent should go, get the product, and pay. It is important to note the simple way to create agents (`createAgent()` method) as well as to communicate with agents (`sendMessage()` method). The reader is referred to [18] for more details on the AgentSpace API.

```
public class BuyerAgentApplet extends Applet {
  ContextView cv = null;
  PlaceView pv = null;
  AgentView av = null;
  ...
  public void init() {
    ...
    cv = AgentSpace.getContextView(...);
    pv = cv.getPlaceOf(...);
  }

  void goAndPay_actionPerformed(ActionEvent e) {
    av = pv.createAgent("BuyerAgent", ...)
    av.start();
    av.sendMessage(Message("goAndPay", "arguments"));
  }
  ...
}
```

The code below shows the `BuyerAgent` class based on which agents are created. Among other particular aspects it is important to note that `run()`, `handleMessage()`, and `afterBackHome()` are predefined callbacks provided by the AgentSpace framework. These methods are called transparently as the consequence of some previous action (i.e., final methods), respectively: `start()`, `sendMessage()`, `backHome()`. Every callback is (or should be, eventually) specialized by the programmer. Lastly, the agent class also shows the existence of helper methods (e.g., the `pay()` method) that are developed occasionally by the programmer.

```
public class BuyerAgent extends Agent {
  public void run() {
    ...
  }

  public void handleMessage(Message m) {
    if ("goAndPay".equals(m.getKey())) {
      PlaceId pid = new PlaceId(m.getProperty("SellerPlace"));
      ...
      Ticket tck = new Ticket(this);
      moveTo(pid, tck, "pay");
    }

    ...
  }

  public void pay() {
    ... // Code to pay using SET
    backHome(getNativePlaceId(), tck, "atHome");
  }

  public void afterBackHome() {
    System.out.println("Transaction finished!");
    ...
  }
}
```

References

[1] A. Chavez and P. Maes. Kasbah: An Agent Marketplace for Buying and Selling Goods. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, London, UK, April 1996.

[2] D. Chess, C. Harrison, and A. Kershenbaum. Mobile Agents: Are They a Good Idea? In [20].

[3] The COSMOS Consortium. *COSMOS – Common Open Service Markets for SMEs.* Esprit Research Project no. 26,850. `http://www.ponton-hamburg.de/cosmos/`.

[4] Rational Software Corp. *UML - Unified Modeling Language.* Version 1.0, 1997

[5] L. Gong. Secure java class loading. *IEEE Internet Computing*, 2(6), November 1998.

[6] F. Griffel, T. Tu, M. Mira da Silva, and M. Merz. Electronic Contract Negotiation as an Application Niche for Mobile Agents. In *Proceedings of the First International Enterprise Distributed Object Computing Workshop*, Queensland, Australia, October 1997.

[7] L. Hurst. MCK: Mobile Communications Kernel. Daghstuhl Seminar on Mobile Software Agents, Daghstuhl, Germany, October 1997.

[8] VISA International and MasterCard International. *Secure Electronic Transaction (SET) Specification.* Version 1.0, May 1997.

[9] Y. Labrou and T. Finnin. A Proposal for a New KQML Specification. Technical Report TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, USA, 1997.

[10] Iridium LLC. *The Iridium System.* `http://www.iridium.com/`.

[11] G. Necula and P. Lee. Safe, Untrusted Agents Using Proof-Carrying Code. In G. Vigna, editor, *Mobile Agents and Security.* Springer-Verlag, November 1997.

[12] ObjectSpace, Inc. *Voyager Core Package Overview.* 1997.

[13] J. Ordille. When Agenst Roam, Who Can You Trust? In *Proceedings of the First Conference on Emerging Technologies and Applications in Communications*, Portland, USA, May 1996.

[14] K. Rothermel and R. Popescu-Zeletin. *Mobile Agents.* Springer, Lecture Notes in Computer Science 1219, April 1997.

[15] T. Sander and C. Tschudin. Protecting Mobile Agents Against Malicious Hosts. In G. Vigna, editor, *Mobile Agents and Security.* Springer-Verlag, November 1997.

[16] A. Silva, M. Mira da Silva, and J. Delgado. AgentSpace: A Framework for Developing Agent Programming Systems. In *Proceedings of the Fourth International Conference on Intelligence in Services and Networks*, Cernobbio, Italy, May 1997.

[17] A. Silva, M. Mira da Silva, and J. Delgado. Improving Current Agent Support Systems: Focus on the Agent Execution System. In *Java Mobile Agents Workshop of the OOPSLA'97*, Atlanta, USA, October 1997.

[18] A. Silva, M. Mira da Silva, and J. Delgado. An Overview of AgentSpace: A Next-Generation Mobile Agent System. In *Proceedings of the Second International Workshop on Mobile Agents*, Stuttgart, Germany, October 1998. Springer, Lecture Notes in Computer Science 1477.

[19] International Telecommunications Union. *Information Technology – Open Systems Interconnection – The Directory: Authentication Framework*, 1993.

[20] J. Vitek and C. Tschudin. *Mobile Object Systems – Towards the Programmable Internet*. Springer, Lecture Notes in Computer Science 1222, July 1997.

[21] Wabasoft, Inc. *The Waba Platform*. `http://www.wabasoft.com/`.

[22] U. Wilhelm, L. Buttyán, and S. Staamann. On the Problem of Trust in Mobile Agent Systems. In *Symposium on Network and Distributed Systems Security*, San Diego, USA, March 1998.

[23] U. Wilhelm and X. DeFago. Objets Protégés Cryptographiquement. In *Proceedings of RenPar'9*, Lausanne, Switzerland, May 1997.

[24] B. Yee. A Sanctuary for Mobile Agents. In *Proceedings of the DARPA Workshop on Foundations for Secure Mobile Code*, Monterey, USA, March 1997.