A Multi-Agent Architecture for Electronic Payment

Sheng-Uei Guan and Feng Hua
Department of Electrical & Computer Engineering
National University of Singapore
10 Kent Ridge Crescent, Singapore 119260

ABSTRACT

The Internet has brought about innumerable changes to the way enterprises do business. An essential problem to be solved before the widespread commercial use of the Internet is to provide a trustworthy solution for electronic payment. We propose a multi-agent mediated electronic payment architecture in this paper. It is aimed at providing an agent-based approach to accommodate multiple e-payment schemes. Through a layered design of the payment structure and a well-defined uniform payment interface, the architecture shows good scalability. When a new e-payment scheme or implementation is available, it can be plugged into the framework easily. In addition, we construct a framework allowing multiple agents to work cooperatively to realize automation of electronic payment. A prototype has been built to illustrate the functionality of this design. Finally we discuss the security issues.

Keywords: Electronic payment, Mobile agent, Electronic commerce, Cryptography

1. INTRODUCTION

1.1 EXISTING PROBLEMS IN E-COMMERCE

The exponential development of the Internet has changed the way enterprises do business. Electronic commerce is becoming an attractive means for conducting business transactions. However, the progress of e-commerce seems to be hindered by the lack of a widely accepted payment standard suitable for e-commerce.

Meanwhile, another factor stymieing electronic commerce is also emerging to the surface. That is lack of intelligence. The vast size of information on the Internet also means that it is difficult for potential customers to locate products that they are interested in. Therefore, e-commerce demands advanced technologies as support. Agent technology seems to be an excellent candidate with its properties of intelligence, autonomy, and mobility. Agent based e-commerce has emerged and become the focus of the next generation of e-commerce [18]. In this new approach, software agents act on behalf of customers to carry out delegated tasks automatically. They support a natural merging of object orientation and knowledge based technology to facilitate reasoning and learning.

1.2 MOTIVATION

Electronic commerce is growing at a tremendous pace, generating a market need for payments of all types. Currently, there are multiple payment schemes like credit card based systems (SET protocol [12]), electronic cash (DigiCash [4]), or electronic checks available for a user. The diversity of payment mechanisms is beneficial since it will create a broader spectrum for exploration of solutions. The needs for diverse payment mechanisms could also be driven by user needs. For example, people pay by cash, check or credit card. Online buyers may also choose their preferred payment methods to complete transactions under different circumstances. Therefore, a system allowing buyers to use different payment methods rather than a single one will give the buyers more flexibility.

A lot of research has been carried out to study how to automate the purchase process. For instance, research on how to automate the process of finding goods on the web and negotiating better prices has been studied widely [6, 13]. Most research work leverages on agent technology because automation needs intelligence. Agent technology is a candidate for solutions.

Payment is the last stage of the whole e-commerce process. Without automating payment, the whole process is not automated. In case of multiple payment options available for buyers to complete a transaction, intelligence is needed to choose the best payment option. For instance, to a particular buyer, usually credit card is preferred over e-cash for transactions. But if a merchant offers discount for e-cash payment, intelligence is needed to choose e-cash as the payment method. In another example, if the buyer has two credit card accounts and one of them has exceeded the limit, intelligence is needed to choose the appropriate credit card account. Therefore an automated payment system needs intelligence and agent technology can offer a solution.

An agent system may consist of a single or multiple agents. In a multi-agent system, distributed control and cooperation among multiple agents will simplify each agent's modular function, speed up a system's operation. Moreover, multi-agent systems present more fault tolerance, since responsibilities are shared among agents.

We propose a multi-agent architecture for electronic payment in this paper. The objective is to accommodate existing multiple payment methods and future payment methods under a scalable architecture. Another objective is to provide a framework allowing multiple agents to work cooperatively to automate the payment process.

This paper is organized as follows: section 2 covers related research background. Section 3 introduces our multi-agent architecture for e-payment. The overall architecture and the interaction protocol among different entities in the framework are elaborated in details. Section 4 discusses and evaluates our prototype. Section 5 compares our work to related e-payment work. In the end, we conclude this paper and look into future work.

2. BACKGROUND

2.1 BRIEF REVIEW OF CURRENT E-PAYPENT SCHEMES

In general, an e-payment system must exhibit integrity, authorization, confidentiality, and anonymity for security requirements [1]. Additionally, there are some other important characteristics such as interoperability, scalability, etc. Specific systems are designed to meet specific requirements, and how these characteristics are balanced poses a challenge to future development.

Payment systems can be classified in a variety of ways according to their characteristics such as the exchange model (cash-like, check-like or hybrid), central authority contact (online or offline), or hardware requirements (specific or general, etc. For example, based on their exchange model, E- payment systems can be divided into categories as shown in Figure 1.

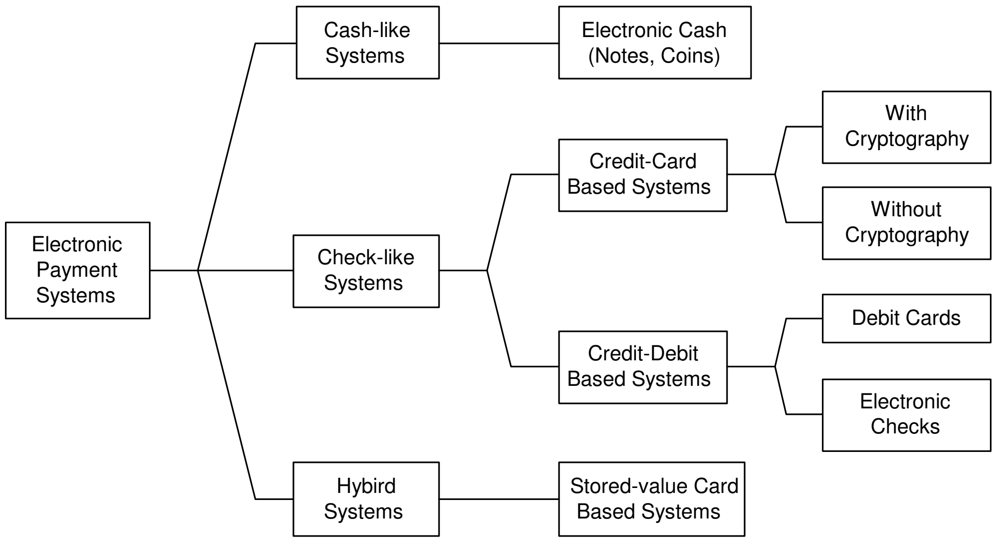

Electronic Payment Systems
Cash-like Systems
Check-like Systems
Hybird Systems
Electronic Cash (Notes, Coins)
Credit-Card Based Systems
Credit-Debit Based Systems
Stored-value Card Based Systems
With Cryptography
Without Cryptography
Debit Cards
Electronic Checks

Figure 1. Classification of Electronic Payment Systems Based on the Exchange Model

The architecture proposed in this paper is built on top of current payment schemes. We give a brief introduction of two typical payment schemes that are adopted as the underlying payment mechanisms in our architecture.

❖ ***Secure Electronic Transaction (SET)***

Currently, a common e-payment method involves a client transmitting to a merchant detailed information of his payment card such as a VISA credit card. This system is simple but susceptible to frauds from either transacting party. The Secure Electronic Transaction (SET) protocol is an evolution of the existing credit-card based payment systems [12]. It provides enhanced security for information transfer as well as authentication of transaction participant identities by registration and certification. It has the potential to become a de facto international standard.

❖ ***Digital Cash (E-Cash)***

Participants of electronic currency payment systems include payers (buyers), merchants, and financial institutions. Digital cash uses electronic token (mostly a unique coded string) to represent monetary value. The bank issuing the tokens has a record of all the tokens. The acquiring bank of the merchants that receive the tokens will transfer them to a clearing house to process them. When the tokens are verified by the issuing bank, the real transaction of funds will take place and the tokens cannot be used again. The usage of digital cash enables full anonymity that cannot be found in other payment systems. Published schemes include E-Cash [2], NetCash, and CAFÉ [10], etc.

2.2.1 RELATED SAFER FRAMEWORK

The proposed e-payment architecture is built in the SAFER context, proposed in earlier research work [5, 18, 22, 23]. We will only give a brief introduction to SAFER. SAFER: **S**ecure **A**gent **F**abrication, **E**volution and **R**oaming, is an infrastructure designed to serve agents in e-commerce and establish necessary mechanisms to manipulate them.

We consider the concept of a mobile agent community which is a basic unit in SAFER. Figure 2 briefly sketches such a SAFER agent community.

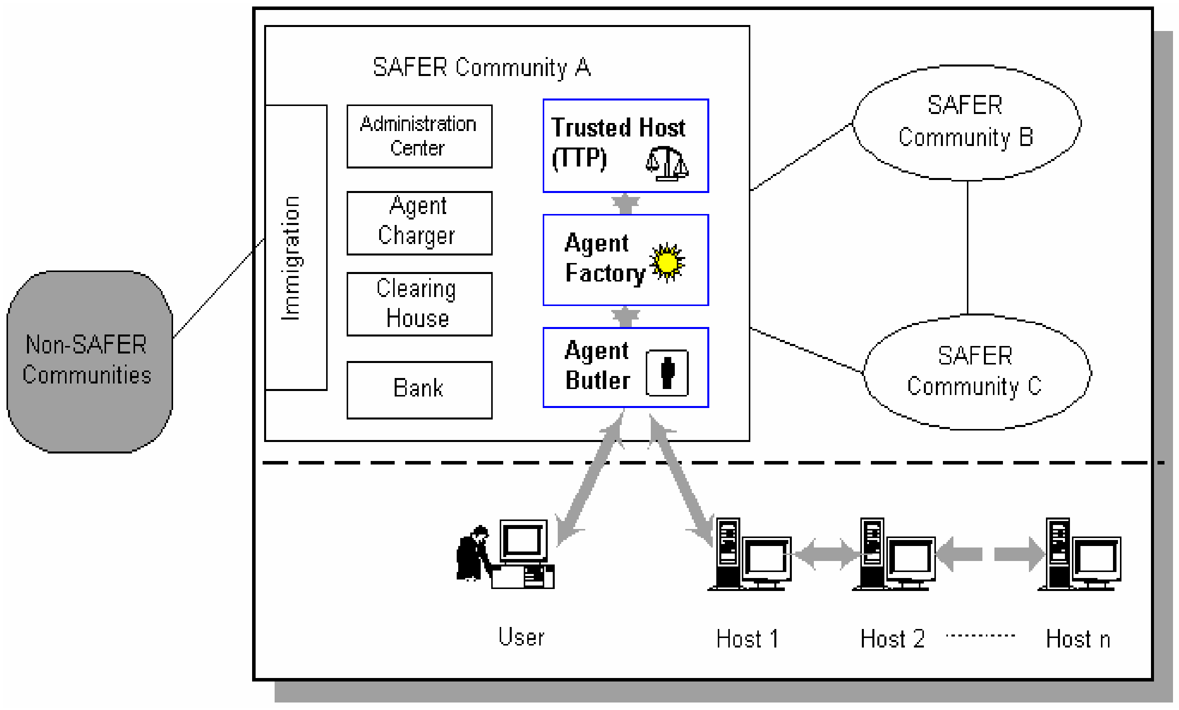


Figure 2. SAFER Agent Communities

As shown in Figure 2, each SAFER community comprises various components and entities. Detailed information of SAFER can be found in [18]. For the clarity of later sections, we briefly introduce several entities involved in our architecture. They are the owner, Agent Butler, Clearing House & Bank and Trusted Third Party (TTP).

The owner doesn't need to be online all the time, but assigns tasks and makes requests to agents via his Agent Butler. Depending on the authorization given, Agent Butler can make decisions on behalf of the owner during his absence, and manage various agents. Clearing House & Bank, as financial institutions in a SAFER community link all value-representations to real money. TTP is a SAFER certified trusted host in a community. Detailed roles of these entities will be discussed in section 3.

Besides those components, mobile agents are the basic units in the framework as well. And in SAFER, it is desirable for agents to have roaming capability. Roaming extends the agent's capability well beyond the limitations imposed by its owner's computer. Mobile agents should be able to physically leave their owners' machines and perform their operations using the computing resources on hosting machines. Details of a mobile agent transport protocol definition can be found in [16].

The payment architecture is considered as an integral part of SAFER hosting and organizing multiple agents to realize automation of electronic transactions.

3. DESIGN OF THE MULTI-AGENT ELECTRONIC PAYMENT ARCHITETURE

3.1 OVERALL NETWORK ARCHITECTURE

In this section, we present the overall network picture of the payment architecture, which contains necessary SAFER components involved in an e-payment transaction.

As shown in Figure 3, there are five major entities in a typical electronic payment transaction. They are Interconnected Financial Institutions, Trusted Third Party, Payment Gateway, Online Shopping Server (Merchant Host), and Agent Butler (owner). These entities as network nodes construct an architecture in which a realization of electronic payment transaction may happen.

Online Shopping Server represents an online e-commerce host, willing to receive and run agents on its local machine. It possesses product information in a local database for agents to access and extract data. Moreover, Merchant Host interacts with Agent Butler and provides related services in case of e-payment transactions.

Financial Institutions consist of bank servers and clearing houses. As depicted in Figure 3, the Issuer refers to the bank that establishes an account for the owner and issues the payment card or electronic checks to the account. The Issuer guarantees payment for authorized transactions using the payment card in accordance with payment card regulations. The Acquirer is the bank that establishes an account with Merchant Host and processes payment cards or validates authorizations and transactions. Payment is implemented by a payer paying the payee via the Issuer and Acquirer [1]. E-Cash server refers to the bank sever that handles issuing and verification of electronic currency.

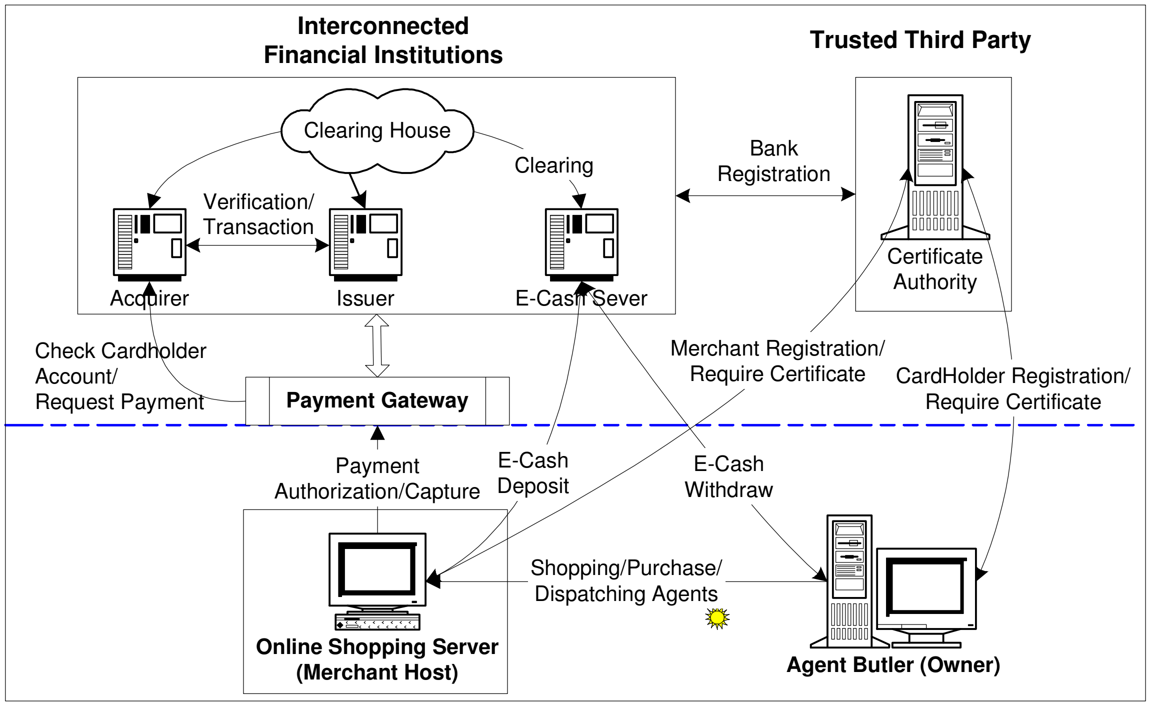

Interconnected Financial Institutions
Trusted Third Party
Clearing House
Clearing
Bank Registration
Verification/ Transaction
Acquirer
Issuer
E-Cash Sever
Certificate Authority
Check Cardholder Account/ Request Payment
Payment Gateway
Merchant Registration/ Require Certificate
CardHolder Registration/ Require Certificate
Payment Authorization/Capture
E-Cash Deposit
E-Cash Withdraw
Shopping/Purchase/ Dispatching Agents
Online Shopping Server (Merchant Host)
Agent Butler (Owner)

Figure 3. Overall Network Architecture

In case that an inter-bank transaction could happen, or different payment forms issued by various banks are adopted by involved hosts, a clearing house will be needed to enable banks to exchange those different e-payment forms with one another and to transfer credits among different SAFER communities. The Clearing House shown in this figure plays this role in the architecture, which facilitates inter-bank transactions especially with large amount of money. Another role is that the Clearing House will be needed to enable credit transactions between banks in SAFER communities and Non-SAFER communities. Since the focus of our work is on consumer-to-business transactions instead of business-to-business transactions, therefore we won't elaborate the Clearing House in further details in this paper.

Payment Gateway (PG) is viewed as the front end of Financial Institution. For example, in a credit-card based system, it works as a device operated by the Acquirer that processes merchant payment messages, including payment instructions from cardholders.

Trusted Third Parties, refer to some neutral SAFER certified trusted hosts in a community. In this paper, the one related to our payment architecture is Certificate Authority (CA). In order to facilitate the provision of security services such as privacy (secure key exchange), non-repudiation (digital signature) and identification, a PKI-based certification module will be used to establish identities for all SAFER entities. Therefore, SAFER entities will be able to identify and authenticate each other in a distributed environment. CA is such a provider of trusted digital certificates for each entity.

Agent Butler resides in a local environment as a static user agent and has a number of functions pertaining to agent management. Agent Butler can dispatch mobile agents to remote hosts. It is responsible for keeping track of agent activities and locations by sending and receiving messages with them. Agent Butler carries out electronic transactions through its Financing Agency (elaborated in the following). In addition, Agent Butler maintains a user interface for interactions with its owner.

3.2 MULTI-AGENT ARCHITECTURE

In Figure 3, we present the whole network architecture. In this section, we will zoom out and focus on the client based multi-agent structure. Under this structure, we use federated multi-agents to accomplish a complex task. This approach suggests that agents should be organized in a hierarchical structure. In a multi-agent e-commerce environment, it is necessary to organize agents into different categories according to their functionalities and competences.

In the architecture, we use “agency” as a subsystem in which a collection of cooperative intelligent agents with specific expertise reside, waiting for tasks from Agent Butler or agency managers. An agency can be regarded as a multi-layered agent group or a federation of agents with specific goal and functional role in the architecture [9]. In other words, it is related to the category of agent classification and organization.

Agencies are under the control of Agent Butler, who helps the owner to keep this virtual environment in order. Under this master-slave design pattern, agents are well organized. Meanwhile, the heavy load and responsibility of Agent Butler is relieved by these well-defined agencies. Therefore, in our architecture, distributed automation and central management are balanced. These agencies interact with each other under the facilitation of Agent Butler and provide services such as information collection, negotiation, decision-making, payment transaction and database maintenance, etc. The agency organization and workflow is depicted in Figure 4. There are Information Agency, Strategy Agency, Negotiation Agency and Financing Agency.

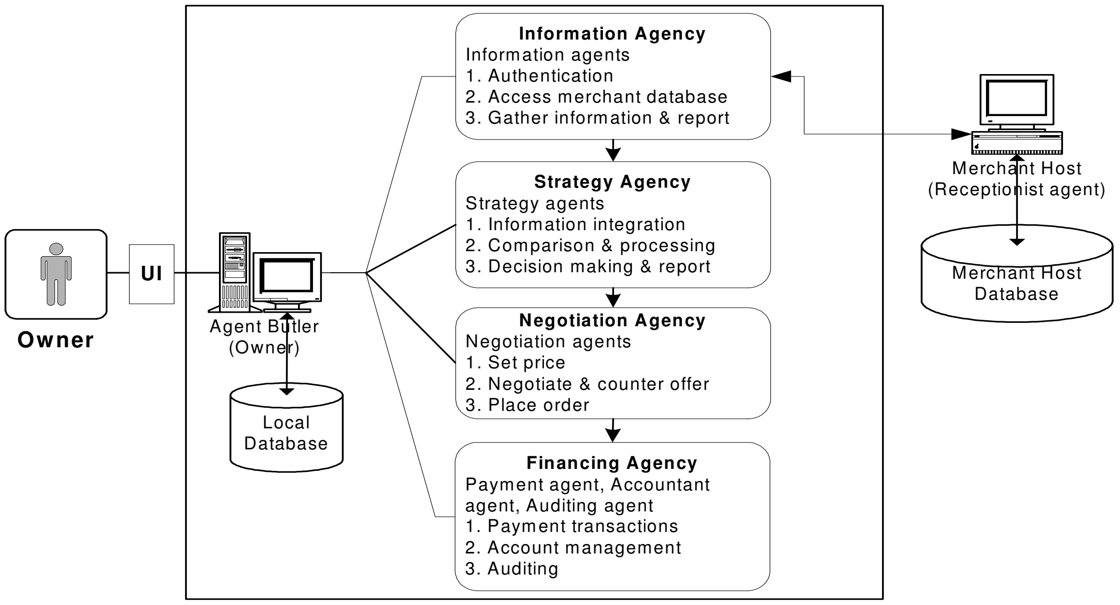


Figure 4. Multi-Agent Architecture

Among these agencies, Financing Agency is the focus in our payment architecture, because only the agents in this agency are involved in payment-related functions. When a purchase decision is made, Agent Butler will activate Financing Agency to initiate a purchase request, and conduct payment in stages via software agents within Financing Agency. More detailed information of Financing Agency and how the agents within this agency collaborate to conduct automated payment transactions will be elaborated in the following sections.

3.3 LAYERED PAYMENT STRUCTURE

In this section, we focus on the hierarchical structure of Financing Agency - a layered payment structure as shown in Figure 5. The payment structure is divided into three layers. They are service layer, interaction layer and payment mechanism layer. This layered design decomposes a complex task into subtasks in which each group of subtasks is aligned to a particular level of abstraction.

The service layer defines a logical layer for different types of services available for the owner, for instance, finance service, information service, etc. Each service is provided by a particular agency. Agency defines the logical mapping for a particular service. If the owner wants a particular service, it can interact with the particular agency.

The interaction layer contains entities that represent the agency to interact with the owner or Agent Butler. Normally these entities are specific agency managers, which control agent service groups (ASGs). A manager is assigned with a specific task by either the owner or Agent Butler, and then it dispatches the task to one of its agents in the ASG it controls. For instance, in order to perform an e-payment transaction, Agent Butler needs to interact with Payment Manager in Financing Agency. Then Payment Manager delegates the e-payment task to one of the payment agents in the payment mechanism layer, referring to some payment scheme. Another example is if the owner needs to register a credit card with Certificate Authority, he interacts with Account Manager in the same agency. Account Manager will then delegate the registration task to the SETRegister agent in the payment mechanism layer. Therefore the owner or Agent Butler does not need to know which agent he is interacting with.

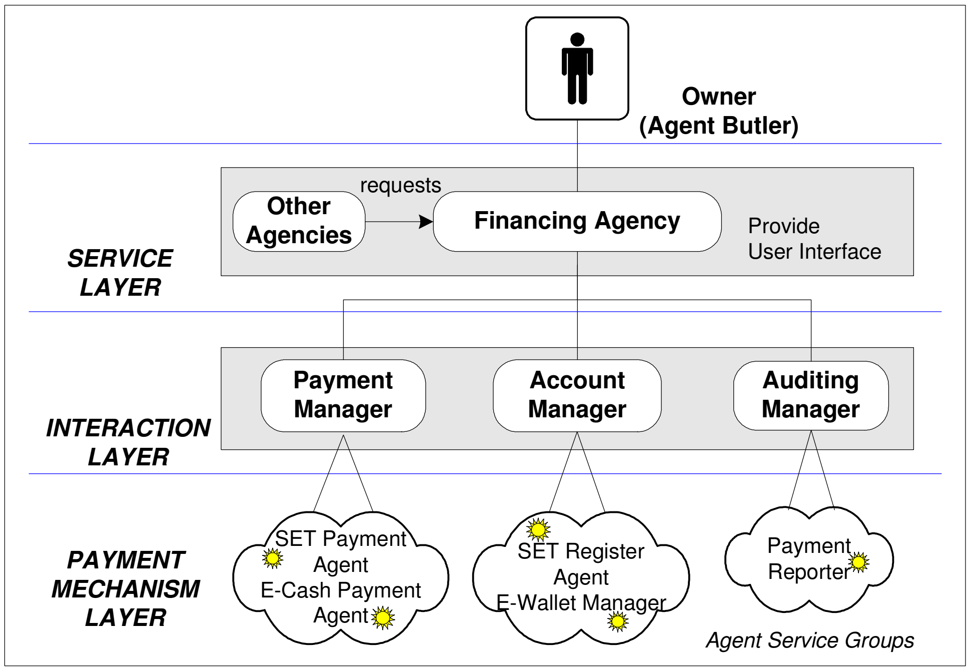


Figure 5. Financing Agency in the Layered Payment Structure

The lowest layer is the payment mechanism layer. It contains agents to perform tasks, for instance, payment via the SET protocol or electronic currency management. In this layer we have Agent Service Groups (ASGs).

Each ASG defines a group of agents that perform similar tasks via different ways. For instance, payment ASG defines a group of agents that are able to conduct e-payment transactions via different protocols, allowing Payment Manager in the interaction layer to manage them easily.

This layered design allows various e-payment schemes to be accommodated into the payment architecture easily, because adding or removing a particular payment agent object in the payment mechanism layer is transparent to the owner or Agent Butler. By defining a uniform interface, agents that implement different payment schemes can be activated by agency managers in the same way.

3.4 ENTITY INTERACTIONS IN SET-BASED E-PAYMENT

By now, we have discussed entities in the architecture and the layered payment structure. In this section, we present how these entities interact with each other to complete an e-payment process using typical payment schemes. We use the SET protocol [12] for credit card based payment as an example to illustrate this process (Figure 6).

In the above entity interaction diagram, we assume a purchase decision has been made, and we proceed now with e-payment. There are two phases during a payment transaction as shown in Figure 6 and particular steps are marked in sequence.

1. Agent Butler receives a purchase decision report from Strategy Agency based on the data collected by mobile information agents.
2. According to pre-defined rules or authorization given by the owner, Agent Butler needs to decide if it should proceed with the payment or it needs to ask for approval or final decision from its owner. This may depend on the authorization given, the amount involved in the payment transaction or some other factors.
3. Once Agent Butler decides to proceed, it delegates the payment task to Payment Manager residing in Financing Agency.
4. Payment Manager invokes an available payment agent through a uniform interface - PaymentAgent (elaborated in section 4) and delegates the payment task to the agent that is able to handle a particular payment method.

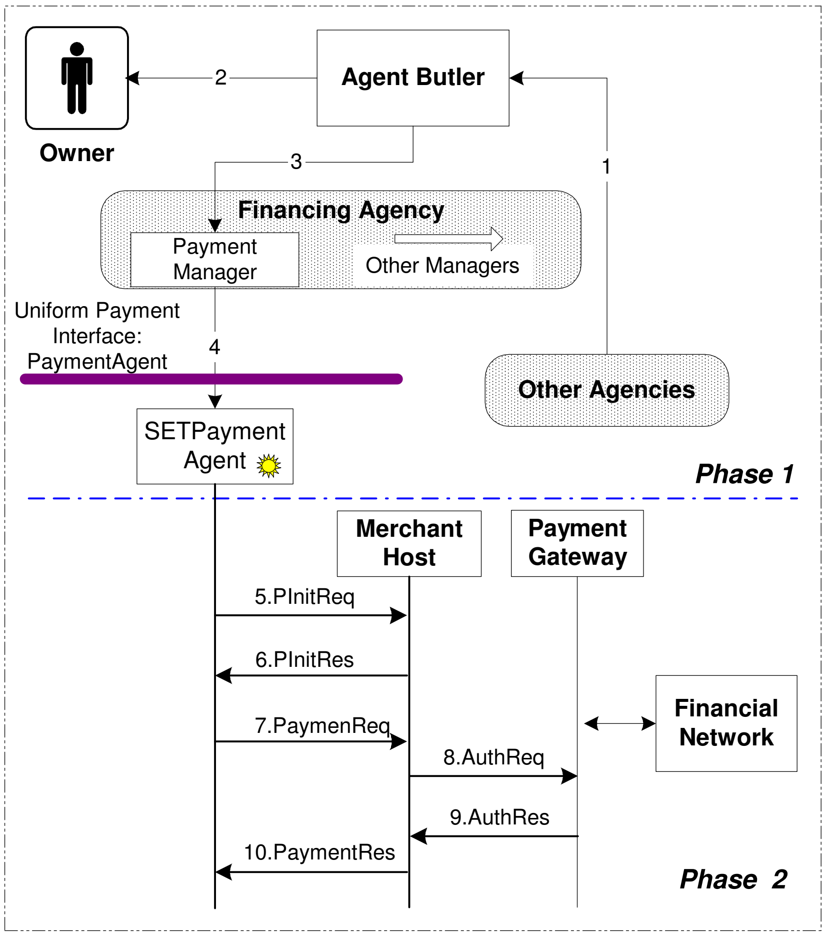


Figure 6. Entity Interaction Diagram for SET Based E-Payment

In step 4, since this e-payment transaction can be settled via different payment methods, e.g. credit card, electronic cash, etc., hence Payment Manager needs to decide which payment method to be used to complete this transaction. This is where intelligence is needed by Payment Manager, which is an agent as well. The decision can be made by Payment Manager, following certain rules set by the owner. These rules can be based on the transaction and payment method information, for instance, the amount to be paid or possible discount if paid by a particular brand of credit card. Based on the information received, Payment Manager chooses a payment method to complete the payment process, and then assigns the task to a related payment agent. In the above entity interaction diagram, we assume credit card payment method is chosen by Payment Manager to complete the payment process, therefore the payment task is delegated to SETPayment Agent which can complete a payment transaction following the SET protocol. And we assume that the SET registration process with Certificate Authority has been done and each entity is issued a set of certificates before a purchase request is made.

During steps 5 to 10 in phase two, SETPayment Agent firstly sends a payment initialization request to Merchant Host. The two parties authenticate each other's identity by exchanging their SET certificates and then SETPayment Agent transmits the encrypted order and payment information to Merchant Host. Merchant Host uses the payment information obtained from SETPayment Agent to make a payment authorization request to Payment Gateway. If these payment instructions are approved, a token is sent to Merchant Host who can make a payment capture request to Payment Gateway using this token later. This would initiate the entire sequence of financial processing at the end of which the actual funds will be deposited into the merchant's account.

3.5 ENTITY INTERACTIONS IN E-CASH PAYMENT

In this section, we use another example of e-cash to show how the architecture accommodates different payment mechanisms. The entity interactions in an e-cash payment process are depicted in Figure 7 and particular steps are marked in sequence.

In phase one, steps from 1 to 4 are similar to what we have discussed in section 3.4. The only difference is that in step 4, Payment Manager assigns the payment task to ECashPayment Agent that can complete a payment transaction following the electronic currency mechanism. Before a payment transaction may happen, there must be enough e-coins stored in the wallet. Otherwise, E-Wallet Manager needs to generate new coins with desired value, and contact the E-Cash bank server, requesting for signing these unauthorized e-coins.

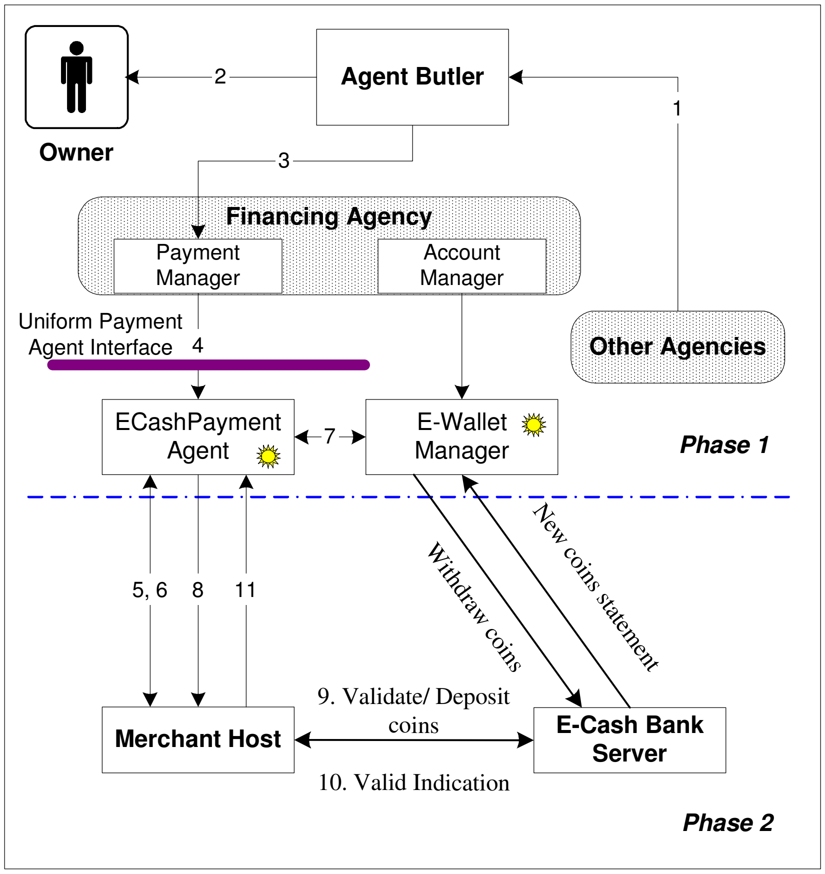


Figure 7. Entity Interaction Diagram for an E-Cash Payment Transaction

5. Having received a payment task, ECashPayment Agent sends a payment initialization message to Merchant Host.
6. Merchant Host sends a payment confirmation message to the client's payment agent. This message contains details about the order amount, the currency to be used, time stamp, and the merchant's bank account ID.
7. After ECashPayment Agent verifies the order information, it checks with E-Wallet Manager and requests for the very amount of e-coins.

8. ECashPayment Agent sends to Merchant Host these e-coins which are encrypted with the bank server's public key.
9. Merchant Host forwards the coins to the bank for validation and deposit.
10. The bank checks these coins are valid and haven not been used before. It sends a valid indication to Merchant Host and deposits the coins into his account.
11. Having received a valid payment, Merchant Host sends purchased items and a receipt to ECashPayment Agent, completing this payment transaction process.

3.6 DISCUSSIONS

In the previous two sections, we have discussed the payment transaction processes using SET and electronic currency as the underlying payment mechanisms. However, a complete e-payment process does not only mean paying money to the merchant, it also includes account management and payment transaction auditing. For each payment protocol, we need a corresponding mechanism to maintain a specific account and keep transaction records in a particular format. For instance, when using the SET protocol, we need to register with Certificate Authority and keep an account which stores sensitive information and maintains personal certificates. When using electronic currency, we also need to maintain an account with the e-cash bank server and manage a local electronic wallet which is used to generate or store e-cash. Therefore, as shown in Figure 5, the interaction layer also includes Account Manager which controls its accounting ASG and delegates tasks to specific account-managing agents. Likewise, there is an auditing mechanism to provide support for recording and maintaining transaction history in case of possible dispute with merchants and financial institutions or enquiry from the owner.

Auditing Manager plays such a role in the architecture by controlling its auditing ASG that may include recording agent or reporter agent.

The tradeoff of this layered design is that efficiency can be sacrificed to some extent, because task invocation is indirect. However the overhead is not significant. The correctness of e-payment is more important and it relies not on task invocation but on payment logic. Besides, efficiency is not among the major concerns but security issues are, which will be elaborated in section 4.3.2.

4. IMPLEMENTATION

4.1 SYSTEM OVERVIEW

The prototype is implemented using the Java programming language. The reason for choosing Java is that Java is an object-oriented and platform independent language, which is suitable for implementing software agents. Besides, Java API includes a security framework, in which various aspects of common security techniques are defined. These security features are used to guarantee valuable data be encrypted when transmitted between different entities over network. Additionally, a 3rd party security provider OpenJCE by an Australian corporation ABA [19] was also adopted. This provider has the necessary support for RSA public-key cryptography which is the main algorithm used in our implementation.

4.2 PROTOTYPE IMPLEMENTATION

4.2.1 Agent Butler

The implementation of the payment architecture began with the simulation of Agent Butler. Agent Butler is able to perform tasks in parallel with its mobile agents. It is responsible for controlling agencies, tracking mobile agent actions, making final decisions. In addition, this stationary Agent Butler provides a GUI to accept data input and display instantaneously to the owner intermediate results of a specific task performed remotely. The modular structure of Agent Butler is depicted in Figure 8.

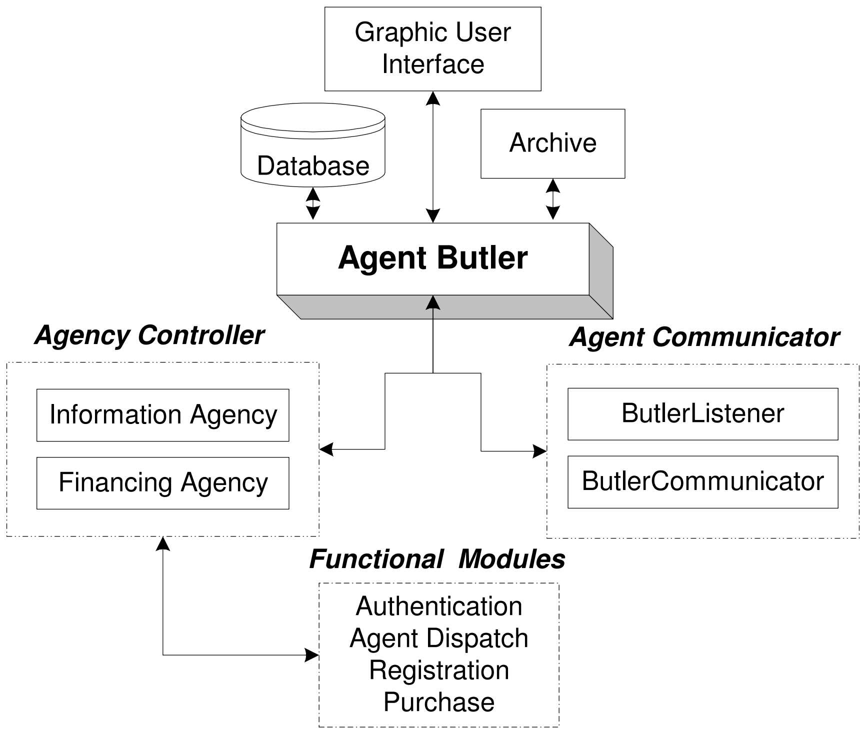
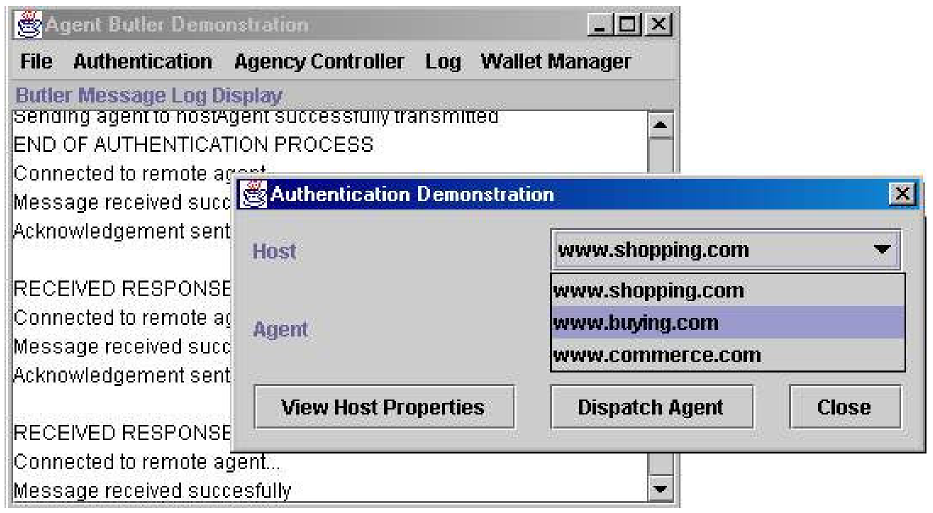


Figure 8. Modular Structure of Agent Butler

As shown Figure 8, Agent Butler has two main function modules, *agency controller* and *agent communicator*. In addition, a database object is owned by Agent Butler and stores information including merchant host information, owner's preferences, etc. Such data may be passed to the related agency objects when needed. *Agency controller* is responsible for activating specific agency according to certain workflow. It is implemented as a member object within the class of Agent Butler. *Agent communicator* handles external socket communications with dispatched mobile agents. *ButlerListener* waits for messages from agents in remote hosts, while *ButlerCommunicator* is capable of sending messages to these agents. Figure 9 shows the trace of the communication process between Agent Butler and a mobile agent to be dispatched.



```
Sending agent to hostAgent successfully transmitted
END OF AUTHENTICATION PROCESS
Connected to remote agent
Message received succ
Acknowledgement sent

RECEIVED RESPONSE
Connected to remote a
Message received succ
Acknowledgement sent

RECEIVED RESPONSE
Connected to remote agent...
Message received succesfully
```

Host: www.shopping.com

Figure 9. Sample Screenshot of Agent Dispatch and Communication

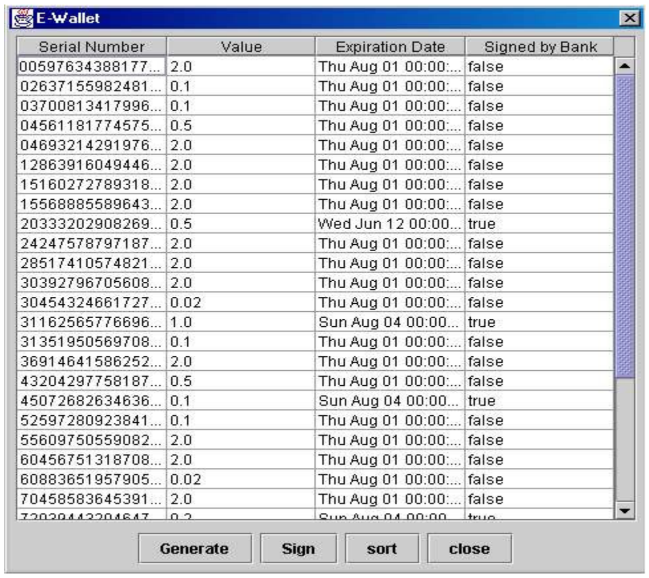
4.2.2 Financing Agency

Financing Agency is implemented as a composite class. It contains a *payment manager object* and an *account manager* object. Composition is a form of aggregation with strong ownership and coincident lifetime as part of the whole. Parts with non-fixed multiplicity may be created after the composite itself, but once created they live and die with it. The two managers are instantiated in the constructor of the *Financing Agency* class. Therefore, they can be accessed via certain access methods (e.g. *getPaymentManager()*) provided by Financing Agency. These two managers are also composite classes, which consist of different agents corresponding to specific payment schemes. In this prototype implementation, two types of payment agents are implemented in Financing Agency. They are *SETPaymentAgent* and *EcashPaymentAgent*. *SETPaymentAgent* is implemented by following the SET protocol. *EcashPaymentAgent* pays by E-Cash, which is defined as shown in Figure 10. *SerialNumber* is simulated as a randomly generated 50-digit numeric string. *Value* denotes the value that this *E-Cash* object represents. *Signed* denotes whether E-Cash has been certified by the bank server. *Expiration Date* denotes the expiration date of E-Cash.

```
public class ECash extends Object
{   private String serialNumber;
    private double value;
    private boolean signed;
    private Date expirationDate;
 }
```

Figure 10. Sample Code of E-Cash

An electronic wallet class is implemented in a local environment, which could be used to manage, generate, and store e-cash. This e-wallet is controlled by E-Wallet Manager belonging to the accounting ASG (Figure 5). When the owner needs some cash, E-Wallet Manager will be activated by Account Manager to interact with the bank server. The screenshot of the e-wallet is shown in Figure 11.



Serial Number	Value	Expiration Date	Signed by Bank
00597634388177...	2.0	Thu Aug 01 00:00:...	false
02637155982481...	0.1	Thu Aug 01 00:00:...	false
03700813417996...	0.1	Thu Aug 01 00:00:...	false
04561181774575...	0.5	Thu Aug 01 00:00:...	false
04693214291976...	2.0	Thu Aug 01 00:00:...	false
12863916049446...	2.0	Thu Aug 01 00:00:...	false
15160272789318...	2.0	Thu Aug 01 00:00:...	false
15568885589643...	2.0	Thu Aug 01 00:00:...	false
20333202908269...	0.5	Wed Jun 12 00:00...	true
24247578797187...	2.0	Thu Aug 01 00:00:...	false
28517410574821...	2.0	Thu Aug 01 00:00:...	false
30392796705608...	2.0	Thu Aug 01 00:00:...	false
30454324661727...	0.02	Thu Aug 01 00:00:...	false
31162565776696...	1.0	Sun Aug 04 00:00...	true
31351950569708...	0.1	Thu Aug 01 00:00:...	false
36914641586252...	2.0	Thu Aug 01 00:00:...	false
43204297758187...	0.5	Thu Aug 01 00:00:...	false
45072682634636...	0.1	Sun Aug 04 00:00...	true
52597280923841...	0.1	Thu Aug 01 00:00:...	false
55609750559082...	2.0	Thu Aug 01 00:00:...	false
60456751318708...	2.0	Thu Aug 01 00:00:...	false
60883651957905...	0.02	Thu Aug 01 00:00:...	false
70458583645391...	2.0	Thu Aug 01 00:00:...	false
72029443204647...	0.2	Sun Aug 04 00:00...	true

Figure 11. Sample Screenshot of the Electronic Wallet

4.2.3 Uniform Payment Interface

As discussed in section 3.4 and 3.5, other payment schemes can be easily implemented and integrated into the architecture via a uniform payment interface - PaymentAgent. The following code in Figure 12 demonstrates how we define this interface.

```
Payment Interface:

public interface PaymentAgent
{   public PaymentInvoice proceedPayment
            (PurchaseInfo [] purchases,
             MerchantInfo merchant);
}

public class PaymentInvoice
{   private ItemInvoice[] itemInvoice;
    private double totalAmount;
}

public class  ItemInvoice
{   private String name;
    private double amount;
    private long purchaseTimestamp;
}

public class PurchaseInfo
{   private String name;
    private int quantity;
    private double unitPrice;
}

public class MerchantInfo
{   private hostName;
    private int portNumber;
}
```

Figure 12. Sample Code of the Uniform Payment Interface

This sample code describes the interface that is implemented by each payment agent object. By defining an interface and forcing each payment agent to implement it, Payment Manager is able to invoke different payment agents in a uniform way. There are more than one payment agent embedded with specific payment mechanism logic in the system. Moreover, decision on which agent to complete the payment task is made during runtime by Payment Manager based on certain rules. Therefore, without defining an interface, Payment Manager has to know each agent to which it delegates the payment task and which method of the chosen agent should be invoked. It makes the system hard to extend because adding a new agent requires adding the logic of invoking this agent in the Payment Manager class.

Since payment agents that implement different payment schemes are invoked via this uniform payment interface, therefore the parameters of the interface methods must provide enough information to allow agents to carry out their payment schemes. To carry out a payment scheme, an agent needs to know what to buy as well as where to buy. As shown in Figure 12, the *PurchaseInfo* class contains information regarding what to buy and the MerchantInfo class contains information regarding where to buy. They are passed as arguments into the method *proceedPayment()*. The *PaymentInvoice* class returned by *proceedPayment()*contains the details of each item purchased. It is recorded and can be used by Auditing Manager for later usage. Payment Manager delegates the payment task by selecting a payment agent and then invokes the *proceedPayment()* method implemented by the payment agent.

4.2.4 Payment Configuration

A configuration file is also needed to define which payment methods are available in the system. By editing the configuration file, a user is able to add or remove payment methods easily. The parameters defined in configuration file are listed in the following:

- *NumberOfPaymentMethods*: It indicates how many payment methods are defined in the configuration file.
- *PaymentMethod(n)Name*: It defines the name of the n-th payment method. The value is a string, which represents the payment method.

- *PaymentMethod(n)Impl*: It defines the implementation of the n-th payment method. The value is a string, which denotes the class name of the payment method implementation. An object of this class is instantiated to execute the payment method.
- *PaymentMethod(n)On*: It flags whether this payment method is activated by the system. The value can be *true* or *false*. *True* means the system is able to activate this payment method. *False* means although the payment method is defined in the configuration file, the system has not activated it.

The configuration file is read in during system startup. A GUI is provided to allow the user to select and activate a payment method. The name of each payment method defined in the configuration file (*PaymentMethod(n)Name*) will be shown on the display. Once the user selects a payment method, a payment object of its implementation class specified in the configuration file as *PaymentMethod(n)Impl* is created. Since all payment method classes implement the uniform payment interface, so a payment process can be activated by invoking the interface implemented by the corresponding payment object. We elaborate with a sample of the configuration file as shown in Figure 13.

```
NumberOfPaymentMethods=2

PaymentMethod1Name=SET
PaymentMethod1Impl=edu.nus.cnn.epayment.set.SETAgent
PaymentMethod1On=true

PaymentMethod2Name=ECash
PaymentMethod2Impl=edu.nus.cnn.epayment.ecash.ECashAgent
PaymentMethod2On=true
```

Figure 13. Configuration File Sample 1

The sample in Figure 13 defines two payment methods. One is based on the SET protocol and the other is based on the E-Cash payment scheme. The class that provides the implementation of the SET protocol is *edu.nus.cnn.epayment.set.SETPaymentAgent*. The class that provides the implementation of the E-Cash payment scheme is *edu.nus.cnn.epayment.ecash.EcashPaymentAgent*. Accordingly, "SET" and "ECash" are loaded into the GUI selection list from which the user chooses a payment method. The user can either choose "SET" or "ECash" as the payment method. If the user chooses "SET", an object of class *edu.nus.cnn.epayment.set.SETPaymentAgent* is created. The implemented uniform payment interface is invoked on that *SETPaymentAgent* object, so that payment can be executed via the SET protocol. If the user chooses "ECash", an object of class *edu.nus.cnn.epayment.set.ECashPaymentAgent* is created. The implemented uniform payment interface is invoked on that *ECashPaymentAgent* object and payment can be executed via the E-Cash payment scheme.

4.2.5 Automated Payment

To automate the whole payment process, we have incorporated a rule-based decision capability into Payment Manager to automate the decision process of choosing a payment agent. A simple scheme is suggested in our architecture. A set of rules is defined in a rule-base for choosing a specific payment method under certain conditions. The template of a rule base is shown in Figure 14.

```
NumberOfRules=n

Rule(n)Priority =
Rule(n)Factor =
Rule(n)Condition =
Rule(n)PaymentMethodName =
```

Figure 14. Rule Base Template

NumberOfRules specifies how many rules are defined in the rule base. In the template, each rule owns a unique ID, which is marked as “(n)” in the above figure.

Additionally, each rule has four attributes, namely Priority, Factor, Condition, and PaymentMethodName. The meaning of each will be clear after we go through the following example.

```
Rule1Priority=1
Rule1Factor= cash-discount & transact-amount
Rule1Condition= (credit-card-discount is true)&(transact-amount
< 50)
Rule1PaymentMethodName=ECash

Rule2Priority=2
Rule2Factor= transact-amount
Rule2Condition=transact-amount < 100
Rule2PaymentMethodName=SET
… … …
```

Figure 15. Rule Base Sample

We have incorporated a rule-based decision facility into Payment Manager to automate the decision process of choosing a payment agent. A simple scheme is included in our architecture. A set of rules is defined in a rule base for choosing a specific payment method under certain conditions. Each rule has one factor that specifies the selection condition with certain priority denotation. Rules are validated in the priority order. Once a rule is valid, the corresponding payment method is chosen. A sample rule base is shown in Figure 15.

The rule base sample defines some rules of selecting a payment method. The first rule has the highest priority 1. The decision factor is cash discount and transaction amount. This rule is valid provided that there is a discount offer for a cash payment and the transaction amount is less than \$50. The second rule has a lower priority 2. The decision factor is transaction amount. This rule is valid provided that the transaction amount is less than \$100. Payment Manager evaluates all the rules defined in the rule base in the priority order. Payment Manager checks whether the condition of the first rule is met. If met, Payment Manager selects ECash as the payment method. Otherwise, Payment Manager continues to evaluate the next rule. If all rules are invalid, Payment Manager can report to the owner and wait for his decision.

4.3 EVALUATION AND DISCUSSIONS

4.3.1 Flexibility

The rationale of designing a multi-agent electronic payment architecture is to provide a framework, which can accommodate different types of payment methods. The prototype is built to illustrate this rationale. The purpose of this prototype is not to build a full-fledged e-payment system, rather it builds a foundation to allow a full functional e-payment system to be built incrementally on top of it. Therefore the focus of the prototype is to define a uniform payment interface, build two types of e-payment agents (i.e. SET and E-Cash) and incorporate the encryption/decryption capability to guarantee a secure transaction.

```
NumberOfPaymentMethods=3

PaymentMethod1Name=SET
PaymentMethod1Impl=edu.nus.cnn.epayment.set.SETAgent
PaymentMethod1On=true

PaymentMethod2Name=ECash
PaymentMethod2Impl=edu.nus.cnn.epayment.ecash.ECashAgent
PaymentMethod2On=true

PaymentMethod3Name=ECheck
PaymentMethod3Impl=edu.nus.cnn.epayment.echeck.ECheckPaymentAgent
PaymentMethod3On=true
```

Figure 16. Configuration File Sample 2

In the following, we demonstrate how a new payment method can be plugged into the system easily. For instance, another payment method *ECheck* is implemented by the class *edu.nus.cnn.epayment.echeck.ECheckPaymentAgent*. To add it into the system, we only need to change the configuration file as shown in Figure 16.

After the system is restarted, the new configuration file is read. Then, the additional payment method *ECheck* is available from the GUI selection list. If the user chooses *ECheck*, an object of class *edu.nus.cnn.epayment.set.ECheckPaymentAgent* is created. The uniform payment interface implemented by *ECheckPaymentAgent* is invoked and payment can be executed via the *ECheck* payment scheme.

To remove or disable a payment method, the user can either remove the entry from the configuration file or simply turn that payment method off by setting "PaymentMethod(n)On=false".

4.3.2 Security

Security is one of the most important issues in electronic payment transactions, e.g. secure data transmission. In mobile agent computing, the security issues also include the security of mobile agents against dishonest shopping servers and the security of shopping servers against malicious agents.

First of all, each entity has to register with Certificate Authority for a set of personal certificates, which are used to enable secure data transmission and to authenticate its identity before any interaction may happen. When mobile agents are sent out to a remote merchant host to collect useful product information, the host always requires adequate authentication proof before accepting further interaction with an agent. Agent Butler is regarded as the owner's legal representative and provided with the owner's certificate and a signature on the Agent Butler's public key and identification number. Before a mobile agent is sent out, Agent Butler sends its authentication proof to the host and requests for a permission token which will be used later as part of the mobile agent's identity validation. Therefore, the host will only accommodate mobile agents from users it trusts.

On the other hand, since Agent Butler is a static user agent staying on the owner's host, thus its integrity is guaranteed. Before sending out a mobile agent, Agent Butler also needs to check the authenticity of a merchant host by requesting for its certificate. In our system, mobile agents are not embedded with payment functionalities, therefore they do not carry sensitive information such as payment information when roaming on the Internet. This would decrease the possibility of its being attacked by malicious hosts. However, the product information gathered from previous merchant hosts may be a target of malicious hosts, therefore, there should be some measure to protect and verify the integrity of mobile agents. Since it is not the focus of this paper, we won't discuss this issue in details. Related research work has been carried out by the SAFER research group [14, 20].

When e-payment transactions are in progress, in order to ensure the integrity of various messages sent and received during an e-cash or SET transaction, all messages are digitally signed with the originating entity's private key. The receiving entity would be able to use the sender's public key from its key-exchange certificate to verify that the message has not been tampered with as well as authenticate the identity of the sending party. To protect the information in the messages from being exposed to unauthorized parties, the message contents are also encrypted whenever possible using a combination of both symmetric and public-key encryption techniques.

4.3.3 Performance Analysis

The objective of the performance testing is to measure how long it takes to complete a payment transaction and to analyze the system performance. We benchmarked the two types of payment methods implemented in our system: SET-based credit-card payment and E-cash payment.

❖ **Credit-card based payment performance result**

The average performance result shown in Table 1 was concluded from more than 10 tests. The average period of time that one credit-card payment transaction takes is 1300 milliseconds. Each credit-card payment transaction consists of four main steps which were also benchmarked in Table 1.

Table 1 Credit-card Transaction Average Performance Result

Total Transaction time (ms)	Transaction Initiation (ms)	Merchant Certificate (ms)	Data Encryption (ms)	Payment Confirmation (ms)
1300	10	160	60	1070

❖ **E-cash payment performance result**

As shown in Table 2, the average time of an E-cash payment transaction is 630 milliseconds. Each E-cash payment transaction consists of several main steps which were also benchmarked in Table 2.

Table 2 E-cash Transaction Performance Breakdown

Total Transaction time (ms)	Transaction Initiation (ms)	Merchant Certificate (ms)	E-cash Withdraw (ms)	Data Encryption (ms)	Payment Confirmation (ms)
630	10	160	10	60	390

Based on the testing results discussed above, we compare the performance of the two payment methods. We notice that the SET protocol based credit card payment takes longer processing time than E-cash payment. Specifically, we find out that the difference mostly exists in the last step, payment confirmation, where the real payment transaction happens at the Merchant side. It takes 1070 milliseconds to finish the step for credit card payment method and 390 milliseconds for E-cash payment method.

However, this difference is reasonable and also expectable in our design. Most of the time costs are spent on message exchanges among different entities as well as the encryption/decryption processing. The SET protocol aims to provide more secure guarantee for e-payment by separating the communication only to related parties in certain steps of the payment process and encrypting all the messages exchanged. In the last step, the Owner, Merchant Host, Payment Gateway (PG) and Certificate Authority (CA) are all involved in message exchanges. In addition, PG and CA are requested to validate the Owner's payment information (related to the Owner's account) before the Merchant can send out payment confirmation to the SET payment agent. The whole process is time consuming. In comparison, E-cash payment is simpler. When the Merchant receives the E-cash notes, it only needs to contact the E-cash bank server to deposit the E-cash. The bank server will validate E-cash. If all the E-cash notes are valid, the bank will send a deposit confirmation to the Merchant who will send out the payment confirmation to the E-cash payment agent. E-cash payment is more efficient than SET-based credit card payment. However, the E-cash bank server needs to validate the E-cash notes one by one. When there is a large amount of E-cash being used, the processing time for E-cash payment will increase accordingly. Therefore, from these facts and analysis, we highly recommend using E-cash payment in small-amount transactions for efficiency and cost saving concern, and using SET-based credit card payment in large-amount transactions.

4.3.4 Discussions

Agent Butler represents its owner and is responsible for agent management and manipulation. When the owner wants to buy some products from some merchant hosts within the SAFER community, he authorizes his Agent Butler to dispatch mobile agents to collect useful information on his behalf. This is done as follows. Firstly, the owner issues Agent Butler a set of encryption keys ($SK_{Ao,}$ PK_{Ao}). Then, the owner authorizes Agent Butler as his legal representative by providing it with his digital certificate (registered with CA) and signing the ID of Agent Butler. In addition, the owner also provides his shopping requirements and a list of trusted merchant hosts' URLs to Agent Butler.

Mobile agents in this prototype do not have the functionality or authority to carry out electronic payment transactions on its own. At this stage, it is still difficult to safeguard agents dispatched to external entities. Vital payment information is thus retained in Agent Butler where it can be easily secured. Transactions are tightly controlled by Agent Butler via its Financing Agency. In the future, when a certain level of integrity and secrecy can be achieved for mobile agents, payment functions could then be incorporated into them.

Compatibility issues arise when integrating different agents into e-commerce websites for transactions. It is suggested that the use of an agent based virtual marketplace could be a stopgap solution before certain protocol standards could be imposed. Virtual marketplace, also under research, considered to be an integral part of SAFER application would also bring with it enhanced security features. Recent work on Semantic Web [8] may offer a long-term solution.

5. RELATED WORK

One research project called BABSy [11] proposed by Rockinger, et al. is also based on the consumer buying behavior model listed above. It is claimed to be an accounting system that helps automated payment in an agent based e-commerce environment. In BABSy, there are only three types of agents which represent the three parties involved in an e-commerce transaction: merchant, bank and user. They are service agent, accounting agent and user agent.

BABSy does not provide a flexible framework that allows more payment mechanisms to be added in future, since adding a new payment method requires modifying the whole user agent. In addition, this approach does not facilitate reusability, since all functionalities are encapsulated inside a single agent of each party.

Research work of an Agent-based Bill Payment Service (ABPS) [15] is also conducted at Queensland University. Their system is hosted on a website which is certified digitally. Consumers must first register with ABPS by providing their personal information. To acquire services, customers authorize an ABPS payment agent to pay the related parties. In their system, the payment agent is responsible for obtaining settlement instructions and settling bills via appropriate financial institutes or external payment services.

ABPS is also centralized to some extent. Except for the payment agent in ABPS, software agents are not explicitly used by participants in their systems. The heavy burden of managing an ever-increasing knowledge base and the growing load for the single payment agent server will be a problem. Our payment scheme avoids a centralized architecture. Instead, we adopted a master-slave design pattern, making use of coopereative multi-agents. Different types of agents are clearly defined and are embedded with certain functional modules as well as decision-making logic according to their function level and roles in the system.

Project Eleanor [25] is an Identrus initiative to introduce secure, direct business-to-business payments on the Internet. Project Eleanor aims to provide Web-based specifications to initiate B2B payments on traditional bank systems. Project Eleanor includes six B2B e-payment options, including payment orders, conditional payment orders, etc. Trading partners will have pre-established instructions with their banks for payment authorization, routing and settlement.

The focus of Eleanor is corporate users and financial institutions. Our payment architecture is to provide business-to-consumer payment solutions. Eleanor is not designed to support a flexible set of payment options that can be easily plugged in. It is more like a clearing-house, or a third party that handles bank-to-bank transactions.

As a different example, the IBM Multi-payment Framework (MPF) offers a suite of software products enabling merchants to use multiple types of payment in Internet commerce [24]. The kernel of the framework is implemented in the IBM WebSphere Payment Manager which is an electronic cash register for merchants. This is an offering for service providers to host payment for multiple remote merchants. It allows merchants to receive payments from consumers on the Internet and to process those payments with banks and financial institutions. Their system enabled merchants to provide or utilize as many payment mechanisms as the customers may need.

The objective of MPF is to provide the capabilities to support multiple payment options for merchants. Therefore merchants in their system are able to deal with consumers who pay in a way that may be different from each other. Our payment architecture is to allow consumers to be able to use different payment methods to pay when they deal with different merchants. MPF and our payment architecture both address the issue of bridging different payment methods between merchants and consumers, but from different perspectives. MPF addresses the problem from the merchant's perspective by providing multiple payment capability on the merchant side. Our system addresses the problem from the consumer's perspective by providing multiple payment capabilities on the consumer side. These two systems should complement each other to provide the greatest flexibilities to all entities involved in e-commerce.

In terms of design, MPF has a similar approach as our agent based payment architecture. It has a layered design and some common interfaces. So different payment methods can be plugged in easily. But their framework does not provide intelligence to choose the best payment option from the merchant's view. In contrast, our system has the capabilities to automatically choose the best payment option for the consumers by using agents based on defined rules.

6. CONCLUSIONS

In this paper, we proposed a multi-agent payment architecture for e-commerce applications. The goal of this architecture is to allow the coexistence of a variety of payment mechanisms and to provide support for multiple agents to collaborate. To achieve these objectives, we adopted a layered design that decomposes a task into subtasks, in which each group of subtasks is aligned to a particular level of abstraction. Payment functionality is fulfilled in Financing Agency, in which, related payment agent objects will handle the details of payment. Users can be relieved from the details of various payment mechanisms. The clearly defined payment interface also facilitates the addition of new payment modules easily. Therefore, the architecture shows greater flexibility and scalability than existing approaches.

REFERENCES

[1] Asokan, N. and Janson, P.A. (1997). The State of the Art in Electronic Payment Systems. Computer, 30 (9), pp. 28 –35.

[2] Brands, S. (1995). Electronic Cash on the Internet. Network and Distributed System Security, 1995. Proc. of the Symposium, pp. 64 –84.

[3] Chavez, A. and Maes, P. (1996). Kasbah: An Agent Marketplace for Buying and Selling goods, in Proceedings of First International Conference on Practical Application of Intelligent Agents and Multi-Agent Technology, London, pp. 75-90.

[4] DigiCash: DigiCash brochure. Available at http://www.digicash.com

[5] Guan, S.U. and Yang, Y. (1999). SAFE: Secure-Roaming Agent For E-Commerce, 26th International Conference on Computers & Industrial Engineering, Australia.

[6] Guttman, R.H. and Maes, P. (1999). Agent-Mediated Negotiation for Retail Electronic Commerce, in Agent Mediated Electronic Commerce. First International Workshop on Agent Mediated Electronic Trading. In Noriega, P., and Sierra, C. (Ed.), Springer, Berlin, pp. 70-90.

[7] Hanaoka, G., Zheng, Y. L. and Imai, H. (1998). LITESET: A Light-Weight Secure Electronic Transaction Protocol. Information Security and Privacy – ACISP'98 LNCS, Vol. 1438, Springer-Verlag, pp. 215-226.

[8] Heflin and Hendler, J. (2000). Semantic Interoperability on the Web. Proc. of Extreme Markup Language 2000, Graphic Communications Assoc., Montreal, pp. 111-120.

[9] Hua, F. and Guan, S.U. (2000). Agents and Payment Systems in 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.

[10] Mjolsnes, S.F. and Michelsen, R. (1997). CAFÉ. Open Transactional System for Digital Currency Payment System Sciences. Proc. of the 13th Hawaii International Conference. Volume: 5, pp. 198-207.

[11] Rockinger, R. and Baumeister, H. (2000). BABSy: Basic Agent Framework Billing System. Proc. of the International ICSC Symposia on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce (MAMA'2000), Wollongong, Australia.

[12] SET: The SET Standard Book 1 Business Description, http://www.setco.org/download.html/#spec

[13] Wang, T.H. and Guan, S.U. (2000). An Agent Based Auction Services for Electronic Commerce. Proc. of International ICSC Congress on Intelligent System & Applications, CD #1524-045.

[14] Wang, T.H. and Guan, S.U. (2001). Integrity Protection for Code-on-Demand Mobile Agents in E-Commerce. Accepted, to appear in Journal of Systems and Software.

[15] Wong, O. and Lau, R. (2000). Possibilistic Reasoning for Intelligent Payment Agents. Proc. of the Second Workshop on AI in Electronic Commerce (AIEC, 2000), pp. 1-13.

[16] Yang, Y. and Guan, S.U. (2000). Intelligent Mobile Agents for E-Commerce: Security Issues and Agent Transport, in Electronic Commerce: Opportunities and Challenges. In Rahman, S.M. and Raisinghani, M. (Ed.), Idea Group, PA., pp. 321-336.

[17] Youll, J. (2001). Agent-Based Electronic Commerce: Opportunities and Challenges. Proc. of the 5th International Symposium on Autonomous Decentralized System, pp. 146-148.

[18] Zhu, F.M., Guan, S.U. and Yang, Y. (2000). SAFER E-Commerce: Secure Agent Fabrication, Evolution & Roaming for E-Commerce, in Internet Commerce and Software Agents: Cases, Technologies and Opportunities. In Rahman, S.M. and Bignall, R.J. (Ed.), Idea Group, PA., pp. 190-206.
[19] JCA/JCE Application Programming Interface Overview,
http://www.openjce.org/docs/jce_api_overview.html
[20] Wang, T.H. and Guan, S.U. (2001). Protecting Integrity for Code-on-Demand Mobile Agents in E-Commerce. Proc. of the First International Workshop on Internet Computing and E-Commerce (ICEC'01), San Francisco, California, USA.
[21] Romao, A. and Silva, M. M. D. (1998). An Agent-Based Secure Internet Payment System for Mobile Computing. Trends in Distributed Systems for Electronic Commerce. Proceedings International IFIP/GI Working Conference TREC'98, 1998, pp. 80-93.
[22] Guan, S.U. and Zhu, F.M. (2002). Agent fabrication and its implementation for agent-based electronic commerce, International Journal of Information Technology & Decision Making, 1 (3), pp. 473-489.
[23] Guan, S.U., Zhu, F.M., and Ko, C.C. (2000). Agent Fabrication and Authorization in Agent-based Electronic Commerce. Proceedings of International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce, Wollongong, Australia, 528-534.
[24] IBM WebSphere Payment Manager:
http://www-3.ibm.com/software/webservers/commerce/paymentmanager/
[25] Identrus and its Project Eleanor: http://www.identrus.com/products/eleanor.xml