Analysis of the SET e-commerce protocol using a true concurrency process algebra *

M. Carmen Ruiz, Diego Cazorla, Fernando Cuartero, Juan J. Pardo

Departamento de Sistemas Informáticos
Univ. de Castilla-La Mancha
Campus Univ. s/n. 02071. Albacete, Spain.
{carmenr, dcazorla, fernando, jpardo}@info-ab.uclm.es

ABSTRACT

A formal specification of the purchase phase in the **SET protocol** (Secure Electronic Transaction), an e-commerce protocol by Visa and MasterCard, is presented. It is modelled by means of a *True Concurrency Timed Process Algebra* called **BTC** which takes into account that the (limited amount of) available resources in a system have to be shared by all the processes. We have considered **heterogeneous resources** and we extend the algebra with the intention of representing the actions which use the shared resources and how many resources of each type the system has at its disposal.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: [Performance of Systems]; D.1.3 [**Programming Techniques**]: [Concurrent Programming]; D.3.1 [**Programming Languages**]: [Formal Definitions and Theory]

General Terms

Formal Specification, SET Protocol

Keywords

Formal Specification, Heterogeneous Shared Resources, Performance Evaluation, Process Algebra, SET Protocol

*This work has been partially supported by the MCyT project "Description and Performance of Distributed Systems and Application to Multimedia Systems" (Ref. TIC2003-07848-c02-02) and the JCCM project "Design and Implementation of Efficient Multimedia Systems by using Formal Techniques" (Ref. PAC-03001)

SAC'06 April 23-27, 2006, Dijon, France

1. INTRODUCTION

The advent of Internet led to the beginning of a whole new set of technologies. One such technology that redefined the manner in which business was conducted is Electronic Commerce (e-commerce). E-commerce involves buying and selling of goods, business-to-business exchange of data and trading with the participating entities in remote locations. Soon, business through e-commerce was a norm but, at the same time, some serious security concerns appeared due to nearly all the transactions carried out through e-commerce are monetary and involves the flow of money in some form (usually using a credit card). So, a large number of e-commerce protocols was proposed as a method to secure payment card transactions over open networks [22, 7, 15] with the main goals of providing the confidentiality, authenticity, and integrity of payment card transmissions. To do this, e-commerce protocols use a variety of cryptographic techniques, including encryption, digital signatures and certificates.

Most Internet merchants use protocols such as SSL or TLS which keep card details safe from eavesdroppers but do nothing to protect merchants from dishonest customers or viceversa. Customers have to trust merchants and, likewise, merchants have to trust customers. These protocols work satisfactorily while the dishonest person is an outsider but it is a fact that to depend on someone is a high-risk policy nowadays. SET protocol (Secure Electronic Transaction) [15] addresses this situation and when a customer makes a purchase, the SET protocol guarantees the authenticity of the transaction while keeping the customer's account details secret from the merchant and his choice of goods secret from the bank. This feature makes SET be firmly in place and guaranteed by major credit card companies all over the world.

E-commerce protocols, however, are inherently complex and any minor flaw in them could compromise the security of the data these security protocols are intended to protect. This has made it necessary to develop formal methods that allow us to study in depth the correction of the systems modelled.

Among the formal methods, we can find, on the one hand, the algebraic models which are based on languages structured with operators, and on the other, the models based on "graphic representations". However, all of them are based on the concept of abstraction in the sense that every model must only capture the characteristic of the system considered important. In modelling concurrent systems the cooperation between the processes demands more attention because it is usually the most difficult part.

Therefore, the first models abstract many real properties of concurrent systems which are considered non essential to them, such as the duration and structure of actions, the properties of communication networks, the distribution of system components, the number of processors and so on. By leaving these non essential properties aside, it is easier to study the behaviour of processes at a formal level due to a clearer design specification. However, these simplifications have a cost which is the loss of all the information which has been abstracted.

Once the basic models have been thoroughly studied, it is necessary to extend them to deal with these other aspects, usually quantitative ones, which have been lost in the abstraction. Extended models have been proposed to represent aspects such as fairness [5, 21], timed models [18, 16], Markovian timed models [12, 11] and probabilistic models [8, 17, 6].

In [19] a timed process algebra, **BTC** (for Bounded True Concurrency), which is based on CSP [13] is introduced. We extended its syntax in order to consider the duration of actions by means of a timed prefix operator, and the operational semantics was also extended in order to consider the context (number of resources) in which processes are executed. A notion of a timed bag of actions was also introduced to represent the concurrent (simultaneous) execution of processes. This algebra allows us to evaluate the performance of a system, taking into account that the (limited amount of) available resources have to be shared by all the processes. In this first approach to the model we considered only homogeneous resources. By homogeneous resource we mean that all the resources to be shared are of the same type. A process can use any of them in the execution of an action which needs a shared resource. On the other hand, when we deal with heterogeneous resources the system has different kinds of shared resources and, depending on the action to be executed, one or another shared resource must be used. In this paper, we have extended this algebra to consider heterogeneous resources also. The most important consequence of this approach is that if there are more processes than resources then not all of them can be executed simultaneously. A process has to wait until it allocates the resources needed to continue its execution. This means that we can find two kinds of delays in the execution of a process: delays related to the synchronization of processes, and delays related to the allocation of resources. The former is usual in a (theoretical) concurrent context, but the latter is only taken into account if we consider a limited amount of available resources.

This process algebra is based on the notion of *true concurrency* where concurrent behaviour can be distinguished from the interleaving one by considering that $(a|b) \not\equiv (a.b + b.a)$. It means that we can allow more than one action to be executed just at once, while in an interleaving behaviour, just one action can be executed at the same time. Our language consider all cases between the two alternatives, so, from this point of view, our model extends both interleaving and true concurrency based models.

In the literature we find several theoretical models that consider the notion of resource consuming (both homogeneous and heterogeneous) but after a detailed study we have found important differences. Let us look at [9, 10] where a discrete timed process algebra for limited parallelism (CCSLP) is presented. Time is associated to actions by considering special actions t_n, where n represents the consumption of n (homogeneous) resource units. This means that if we want to specify an action lasting 2 units of time, we have to write $a.t_1.t_1.NIL$, where a does not represent the execution of the action but only the moment in which the action starts its execution. From our point of view, the main problem of this model is that it may be difficult to realise when an action finishes. Thus, it is difficult to analyse the performance of a system because it may be impossible to know when the system has reached a certain state.

In [4] a process algebra for shared processors is presented. The paper is focused on processes sharing a single processor scheduled according to some scheduling strategy. Multiprocessors systems can be described, but only with static allocation of processes to processors. This situation does not happen in our model where there is no assignment of processes to processors, i.e., a sequential process may be interrupted, and later on it can continue its execution in a different processor.

ACSR, a process algebraic approach to the specification and analysis of resource-bound real-time systems, is introduced in [14]. In this process algebra, the use of shared (heterogeneous) resources is represented by timed actions and synchronization is supported by instantaneous events. Timed actions are always executed for one time unit (*tick*) and they consume a set of resources during that time. Synchronization between timed actions is allowed but only in the case of both actions having the same duration and using disjointed sets of resources. The approach taken in ACSR and extensions (with probabilities or value passing) is slightly different from ours, mainly because resources are statically assigned to processes and this does not occur in our model, where resources are dynamically assigned to processes.

In the present paper we have extended **BTC** with a view to representing in the specification both the set of actions which need one of the shared resources for their execution and the number of resources at its disposal in the system. We have considered **heterogeneous resources**, so we need to define a different set of actions for each type of shared resource. With this new approach we are able to model more complex real systems relatively easily and moreover, this model is a great help in the design of the system because it allows us to find the minimum number of resources needed to obtain the best performance of the system, and to evaluate the performance of a system when the number of shared resources is fixed.

After reporting this new approach, we will use it to present a formal specification of a realistic system: the Secure Electronic Transaction (SET), which has proved to be particularly resistant to verification. Nevertheless there have been several attempts at verifying it formally; among the best is Paulson and Bella's work [1, 2] but, to our knowledge, SET protocol is still in research.

SET defines a variety of transaction protocols to securely conduct electronic commerce. This family of protocols consists of: Cardholder registration, Merchant registration, Purchase request, Payment authorization and Payment capture, which can be split into two phases: the registration phase and the purchase one. We will leave the bootstrap phase of SET until later and focus our efforts on the **purchase**

phase of SET.

The structure of the rest of this paper is as follows. First, we make an outline of the SET protocol, identify the entities which take part in the purchase phase and define (as clearly as possible) the actions each process must do. In Section 3 and 4 we present the syntax of **BTC** and its operational semantics by means of a set of inference rules. Next, we specify the SET protocol by means of this algebra. We finish with some concluding remarks and our future work.

2. OVERVIEW AND ABSTRACTION OF THE SET PROTOCOL

The Secure Electronic Transaction (SET) protocol has been proposed and standardized by a consortium of credit card companies (VISA and Mastercard) and software corporations (Microsoft, Netscape, etc.) to improve the security of credit card purchases. In this section we do not intend to discuss the advantages and disadvantages of this novel protocol, nor explain its behaviour or specification (more than 1000 pages in its documentation is enough). However we will present an overview of it in the most accessible and simplified way possible.

SET comprises five main protocols:

- Cardholder registration: This allows a cardholder to register with a Certificate Authority (CA) before making a purchase.
- Merchant registration: Like cardholders, merchants must register with a Certificate Authority before they engage in transactions.
- Purchase request: This protocol is invoked after the cardholder has completed browsing, selection and ordering. As is the norm, the cardholder initiates the transfer of messages by sending the order information and payment instructions.
- Payment authorization: The merchant is allowed to verify the cardholder's details with a payment gateway, which authorizes the transactions.
- Payment capture: A payment gateway checks that everything is satisfactory and the actual funds transfer is made.

These five protocols can be split into two phases:

- the registration phase: Getting the approval of the cardholder's or merchant's bank, and replacing traditional credit card numbers with electronic credentials so that their privacy is protected.
- the purchase phase: It captures the purchase request, payment authorization and the payment capture protocols which are at the kernel of the SET protocol.

We will leave the registration phase of SET until later and focus our efforts on the **purchase phase** which involves these parties:

- Cardholder: an authorized holder of a payment card supported by an Issuer and who uses the card in electronic commerce.
- Merchant: provides goods, services and/or information and accepts payment for them electronically.
- Acquirer: a financial institution that supports merchant activity through relations with merchants. It is responsible for gathering the financial data related to the transaction in order to obtain authorization for payment from the cardholder's Issuer.
- Payment gateway : a system that provides electronic commerce services to the merchants in support of the Acquirer, and interfaces with the payment card's financial network to support the authorization and capture of transactions. Payment gateways represent the merchant's financial institution on the Internet.

We consider the Acquirer and the payment gateway to be just one entity (essentially a bank), because for our model the exchange of information between them is not essential.

3. THE LANGUAGE BTC

In this section we introduce the process algebra **BTC** by means of its syntax and show how we have improved it so that we can deal with both homogeneous and heterogeneous resources.

Definition 1. *Let $\mathcal{A}ct_T$ be a finite set of timed actions and $\mathcal{A}ct_U$ a finite set of untimed actions, $\mathcal{A}ct_U \cap \mathcal{A}ct_T = \emptyset$. The syntax of* **BTC** *is defined by the following BNF expression:*

$$P ::= stop \mid a.P \mid <b,\alpha>.P \mid P \oplus P \mid P + P \mid P \parallel_A P \mid recX.P$$

where $A \subseteq \mathcal{A}ct_U$, $a \in \mathcal{A}ct_U$, $b \in \mathcal{A}ct_T$, and $\alpha \in \mathbb{N}$, where $\mathbb{N}$ represents the set of natural numbers. Furthermore, we assume a set of process variables Id ranged over by X, X', a set of processes $\mathcal{P}$ ranged over by P, Q, R, and a set of actions $\mathcal{A}ct = \mathcal{A}ct_U \cup \mathcal{A}ct_T$.

We extend this syntax with a view to representing both the actions that use the shared resources and amount of resources the system has at its disposal:

$$[\![P]\!]_{Z,N}$$

where $Z = \{Z_1,\ Z_2,\ ...,\ Z_m\}$ represents the set that consists of a set for each different type of shared resource. Each of these sets is defined as following: $Z_i = \{b_1,\ b_2,\ ...,\ b_{x_i}\}$ and represents the set of actions which require for its execution at least one of the shared resources of the type the set has been defined. $m \in \mathbb{N}$ is the number of different types of shared resources available in the system and $x_i \in \mathbb{N}$ is the number of actions which need at least one resource of type i for its execution. We consider

$$N = \{N_1,\ N_2,\ ...,\ N_m\}$$

where $N_i \in \mathbb{N}$ represents the amount of shared resources of type i available in the system.

With these changes, our time process algebra **BTC** is able to model processes which need to use different types of resources in their execution (heterogeneous resources), which is an important step towards our goal of being able to model more complex real systems.

Let us now describe informally the interpretation of these operators. Later we will provide them with an operational semantics, so the meaning of each operator and the relationship between them will be completely formalised.

stop. This represents a deadlock; that is, no action can be executed.

Prefix. This is the classic prefix operator. We will use it to represent a process P prefixed by an untimed action. As usual, the process $a.P$, once the action a has been executed, behaves like the process P. Mainly, we will use the (untimed) prefix operator to represent a possible synchronization of a process.

Timed prefix. The classic prefix operator has been enriched with information about the time the action b takes in its execution (the value α). The process $< b, \alpha>.P$, once the action b has been executed, behaves like the process P.

Internal choice. This is essentially the classic internal choice operator. Given two processes P_1 and P_2, $P_1 \oplus P_2$ represents the process that behaves like P_1 or like P_2 as the result of an internal decision of the system.

External choice. This is also interpreted in the usual way. Given two processes P_1 and P_2, $P_1 + P_2$ represents the process that behaves like P_1 or like P_2 as the environment requests.

Parallel composition. $P_1 \parallel_A P_2$ represents the parallel execution of the processes P_1 and P_2, where they synchronize on the actions in A. We will denote by $P_1 \parallel P_2$ the parallel execution of P_1 and P_2 without synchronization, that is, $P_1 \parallel_\emptyset P_2$.

Recursion. The process $recX.P$ represents the classic recursion operator, where occurrences of X are substituted by $recX.P$. This operator allows us to define infinite behaviours.

During the rest of the paper we will use the following notation to deal with multisets (bags). We will use the convention that $B = \{2.x_1,\ 3.x_2,\ 5.x_3\}$ represents the multiset defined over the set $X = \{x_1,\ x_2,\ x_3,\ x_4\}$ by:

$$B : \{x_1,\ x_2,\ x_3,\ x_4\} \longrightarrow \mathbb{N}$$

$$B(x_1) = 2,\ B(x_2) = 3,\ B(x_3) = 5,\ B(x_4) = 0$$

We will use the capital letters $B,\ B_1,\ B_2,\ C,\ C_1,\ C_2$ to denote multisets, while capital letters such us $A,\ A_1,\ A_2$ will represent sets. Both the empty multiset and the empty set will be represented by $\emptyset$. We will also use the following operations:

- $B_1 \ \cup\ B_2$: Union of multisets.
- $|B|_Z$: Number of actions in the multiset which need one shared resource from set Z for their execution, that is, $|B|_Z = \sum_{x \in Z} B(x)$.
- $B \cap A$: Multiset obtained by taking from B the actions belonging to A, that is,

$$(B \cap A)(x) \ = \begin{cases} B(x) & \text{if } \ x \in A \\ 0 & \text{otherwise} \end{cases}$$

Definition 2. *Let $\mathcal{B}(X) = \{B \mid \forall x \notin X, B(x) = 0\}$ be the set of multisets over the set X. A timed bag is a pair (B, α), where $B \in \mathcal{B}(X)$, and $\alpha \in \mathbb{N}$ is the duration of every element in B. The set of timed bags over the set X, denoted by $TB(X)$, is defined as $TB(X) = \{(B, \alpha) \mid B \in \mathcal{B}(X) \wedge \alpha \in \mathbb{N}\}$.*

Intuitively, a timed bag relates a multiset to a unique natural number that represents the time every action (element) in the multiset needs to finish its execution.

4. OPERATIONAL SEMANTICS

In this section we present an operational semantics. By means of this mechanism, we provide the language operators with a meaning and an accurate interpretation by describing how one process is able to turn into another. This evolution is represented by using labelled transition systems.

We will have that *transitions* are triples $(P, Q, (B, \alpha))$. We will usually denote transitions by $P \xrightarrow{B,\alpha} Q$.

Intuitively, the meaning of the previous transition is that the process P executes the actions belonging to the multiset B and then it behaves like the process Q. The execution of each action belonging to the same multiset takes α units of time.

In order to consider internal (non observable) evolutions of a process, we will consider a new kind of action, $\tau \notin \mathcal{A}ct$, which represents an internal action. This action is used to define the evolution of the internal choice of two processes, and it has a duration 0.

In Table 1 we find the rules of the operational semantics. We assume that $\alpha, \alpha' \in \mathbb{N}$, $a \in \mathcal{A}ct_U$, $b \in \mathcal{A}ct_T$, $B_i \in \mathcal{B}(\mathcal{A}ct \cup \{\tau\})$ and $B' \in \mathcal{B}(\mathcal{A}ct_U)$. We also consider a constant N that represents the amount of processors/resources at disposal.

The rule $R1a$ is the classic one for the prefix operator. The rule $R1b$ states that, given a process P and an action b with an execution time α, b is first executed and, after α units of time, the process behaves like P. In rule $R1c$ a partial execution of the timed action b is allowed, i.e., action b is executed for α' units of time and the remaining time $(\alpha - \alpha')$ will be performed later on.

The rule $R1c$ allows a prefix operator transition to be split into any number of consecutive transitions. This rule is very similar to the rule **ActT** presented in [3]. The only difference is that, in the latter case, a dense time domain is considered and, consequently, an infinite number of transitions are derived. In our model this does not occur because we work in a discrete time domain.

The rules $R2$ and $R3$ are the classic ones for the internal and external choice operators.

The rule $R4$ represents the basic synchronization mechanism. We only consider those synchronization actions that may be performed by processes P and Q. As synchronization actions are untimed actions, the whole process evolves in 0 units of time to $P' \parallel_A Q'$.

The rule $R5a$ captures the simultaneous execution of no synchronization actions with the restriction that, for each type of resource, N_i actions, at most, can be executed simultaneously. These are actions which need a shared resource of this type for their execution.

Although we have considered as premises that both processes may perform different timed bags with the same time α, this does not represent a loss of generality because applying rule $R1c$ we can split transitions in order to have the same time in both premises.

The rules $R5b$ and $R5c$ are similar to $R5a$ but consider that only one of the two processes (P or Q) evolves.

Finally, the rule $R6$ captures the semantics for recursion in the usual fashion.

R1a) $$\frac{}{a.P \xrightarrow{\{a\},0} P}$$

R1b) $$\frac{}{(b,\alpha).P \xrightarrow{\{b\},\alpha} P}$$

R1c) $$\frac{0 < \alpha' < \alpha}{(b,\alpha).P \xrightarrow{\{b\},\alpha'} (b,\alpha-\alpha').P}$$

R2a) $$\frac{}{P \oplus Q \xrightarrow{\{\tau\},0} P}$$

R2b) $$\frac{}{P \oplus Q \xrightarrow{\{\tau\},0} Q}$$

R3a) $$\frac{P \xrightarrow{B,\alpha} P'}{P + Q \xrightarrow{B,\alpha} P'}$$

R3b) $$\frac{Q \xrightarrow{B,\alpha} Q'}{P + Q \xrightarrow{B,\alpha} Q'}$$

R4) $$\frac{P \xrightarrow{B',0} P' \wedge Q \xrightarrow{B',0} Q' \wedge B' = B' \cap A \neq \emptyset}{P \parallel_A Q \xrightarrow{B',0} P' \parallel_A Q'}$$

R5a) $$\frac{P \xrightarrow{B_1,\alpha} P' \wedge Q \xrightarrow{B_2,\alpha} Q' \wedge B_i \cap A = \emptyset \wedge |B_1|_Z + |B_2|_Z \leq N}{P \parallel_A Q \xrightarrow{B_1 \cup B_2,\alpha} P' \parallel_A Q'}$$

R5b) $$\frac{P \xrightarrow{B,\alpha} P' \wedge B \cap A = \emptyset}{P \parallel_A Q \xrightarrow{B,\alpha} P' \parallel_A Q}$$

R5c) $$\frac{Q \xrightarrow{B,\alpha} Q' \wedge B \cap A = \emptyset}{P \parallel_A Q \xrightarrow{B,\alpha} P \parallel_A Q'}$$

R6) $$\frac{P\{recX.P/X\} \xrightarrow{B,\alpha} P'}{recX.P \xrightarrow{B,\alpha} P'}$$

Table 1: Operational Semantics

DEFINITION 3. *The operational semantics of* **BTC** *is defined as the least multiset of transitions we can derive by using the rules in Table 1.*

5. THE BTC MODEL FOR THE SET PROTOCOL

As we have said, our focus is on purchase phase of the SET protocol which consists of three protocols: Purchase Request, Payment Authorization and Payment Capture. We now check how they work before presenting our model. Figure 1 provides a high level overview of the Purchase Request protocol, showing its main steps which can be summed up as follows:

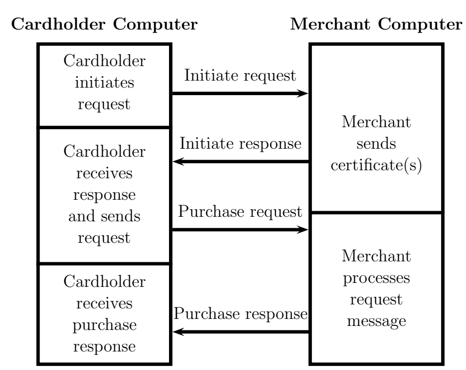

Figure 1: High level overview of the Purchase Request

1. The cardholder (customer) presents a purchase request to the merchant which includes credit card information and an order form.

2. When the merchant receives the request, it assigns a unique transaction identifier to the message.

3. The merchant sends a response along with the merchant and payment gateway certificates to the cardholder.

4. The cardholder verifies the merchant and gateway certificates.

5. The cardholder creates the Order Information (OI) and Payment Instructions (PI). We must point out that the OI does not contain the order data, such as the description of goods; this kind of information is just exchanged between the cardholder and the merchant.

6. The cardholder transmits a message consisting of the OI and the PI to the merchant.

7. The merchant verifies the cardholder certificate and checks the digital signature to ensure the order has not been tampered with in transit and that it was signed using the cardholder's private signature key.

8. The merchant processes the order, including the payment authorization.

9. The merchant transmits information to the cardholder about the reception of the cardholder's order and whether the transaction has been refused or approved. If it has been approved, the merchant will ship the goods or perform the services indicated in the order.

10. When the cardholder receives the purchase response message from the merchant, he verifies the merchant signature and takes action based on the contents of the response message, such as displaying a message to the cardholder or updating a database with the status of the order.

In the previous step 8 the merchant needs to get the transaction authorized, which is done by the payment gateway. In this task two protocols are involved: Payment Authorization and Payment Capture. The first is shown in Figure 2 and can be abstracted in this way:

1. The merchant generates an authorization request which includes the amount to be authorized, the transaction identifier from the OI and other information about the transaction.

2. The merchant sends the authorization request and the cardholder payment instructions to the Payment Gateway.

3. When the Payment Gateway receives the authorization request, it verifies:

   - The merchant certificate.
   - The merchant certificate has not expired.
   - The request was signed using the merchant private key.
   - The cardholder certificate.
   - The cardholder certificate has not expired.
   - The PI has not been tampered with in transit and was signed using the cardholder private key.
   - The transaction identifier received from the merchant matches the one in the cardholder PI.

4. The Payment Gateway sends an authorization request to the Issuer.

5. Upon receiving an authorization response from the Issuer, the Payment Gateway transmits it to the merchant.

6. The merchant stores the authorization response to be used when requesting payment (through Payment Capture protocol)

7. The merchant completes the processing of the cardholder's order (step 9 in Purchase Request protocol).

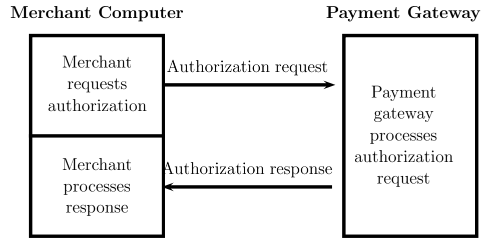

Figure 2: High level overview of the Payment Authorization

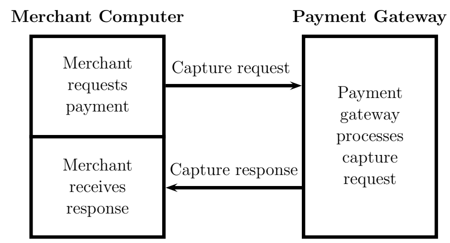

Figure 3: High level overview of the Payment Capture

After completing the processing of an order from a cardholder, the merchant will request payment by means of the last protocol we are going to abstract: the Payment Capture one, whose high level overview is shown in Figure 3 and which we summarize as follows:

1. The merchant creates a capture request with information about the transaction, such as the final purchase amount, and transmits it to the Payment Gateway.

2. The Payment Gateway makes some verifications and sends the appropriate messages that the payment was made.

3. The response is then transmitted to the merchant by the Payment Gateway.

4. The merchant receives the capture response and stores it to be used for reconciliation with payment received from the Acquirer.

Now we have been able to abstract the phase of the protocol which we are concerned with, we can start to make a specification of it in **BTC**. In the system we find working in parallel the following:

- A number (m) of customers (**cardholders**) who have just finished browsing, selection and ordering. Each one must send an initial request to the merchant ($ini\_request_i$) and wait for its response ($ini\_response_i$), where the cardholder receives the merchant and gateway certificates which he verifies (c_verify_i). After this he must create the Order Information and Payment Instructions (OI_PI_i), transmit it to the merchant ($purchase\_request_i$) and wait for a response ($purchase\_response_i$) while the merchant processes the order and determines if it is approved. When the cardholder receives the purchase response message from the merchant, he verifies the merchant signature (c_verify_i) and takes action based on the contents of the response message.

- A **merchant** system which provides each cardholder with a virtual merchant.
- As many virtual merchants $\mathbf{M_i}$ as cardholders. One type of shared resource in our abstraction of the system consists of the real available merchants, so the set Z_1, with the actions which need one of these real merchants for their execution, is made up of no synchronization actions which the virtual merchants must execute:

  $$Z_1 = \{assign\_identifier_i, m\_verify_i, creat\_autho_i\}$$

  A virtual merchant is created to serve a cardholder, so after the $ini\_request_i$ the merchant assigns a unique transaction identifier ($assign\_identifier_i$) and sends it along with the merchant and payment gateway certifier ($ini\_response_i$). When he then receives the OI and PI from the cardholder ($purchase\_request_i$), he must verify it (m_verify_i) and generate an authorization request ($creat\_autho_i$) which is sent to the Payment Gateway ($autho\_request_i$), which makes all the required verifications and obtains an authorization from the Issuer which is transmitted to the merchant ($autho\_response_i$). With this authorization, the merchant is able to complete the processing of the cardholder's order ($purchase\_response_i$) before requesting payment ($capture\_request_i$) through the Payment Gateway.

- A **Payment Gateways** system which provides each merchant with a virtual Payment Gateway.
- As many virtual Payment Gateways $\mathbf{P_i}$ as merchants. The other type of shared resource in the system consists of the real available Payment Gateways, so the set Z_2 is made up of no synchronization actions which need one of these real payment gateways for their execution:

  $$Z_2 = \{p\_verify_i, autho\_issuer_i, pay\_issuer_i\}$$

  Any virtual Payment Gateway is in charge of authorizing transactions when required ($autho\_request_i$) and sending this information to the merchant ($autho\_response_i$). When the Payment Gateway receives the authorization request, it must carry out several verifications (step 3 in the previous abstraction of the system (p_verify_i), and send an authorization request to the Issuer ($issuer\_autho\_req_i$). The Issuer generates the authorization response ($autho\_issuer_i$) and sends it to the Payment Gateway ($issuer\_autho\_res_i$).

  In addition, when the order has been shipped the Payment Gateway process receives the request payment from the merchant ($capture\_request_i$), whose information must be verified (p_verify_i) and sent to the Issuer ($pay\_capture\_req_i$). After the payment has been made (pay_issuer_i), the Issuer sends the information to the Payment Gateway ($pay\_capture\_res_i$) which sends the capture response to the merchant ($capture\_response_i$), to be added later to the payment received from the Acquirer.

As we have said, we have two types of shared resources, one being the real available merchants that defines the set Z_1 with the actions which need one of these shared resource for its execution. The other type consists of the real payment gateway which defines the set Z_2 in the same way. Later on, when we know the amount of available resources of any type in the system we are trying to model, we will set the value of the variables N_1 and N_2.

$$Z_1 = \{assign\_identifier_i, m\_verify_i, creat\_autho_i\}$$
$$Z_2 = \{p\_verify_i, autho\_issuer_i, pay\_issuer_i\}$$

Therefore, the specification of the system in **BTC** is as follows:

$$\begin{aligned}
&[\![\text{SET_System}]\!]_{Z,\ N} \equiv [\![cardholder_1 \parallel \ldots \parallel cardholder_i \parallel \ldots \parallel \\
&\qquad cardholder_m \parallel merchant \parallel \\
&\qquad pay_gateway_1 \parallel \ldots \parallel pay_gateway_s]\!]_{Z,\ N} \\
&cardholder_i \equiv ini_request_i.ini_response_i.(c_verify_i, 1). \\
&\qquad (OI_PI_i, 3).purchase_request_i. \\
&\qquad purchase_response_i.(c_verify_i, 1).cardholder_i \\
&merchant \equiv M_1 \parallel M_2 \parallel \ldots \parallel M_i \parallel \ldots \parallel M_m \\
&M_i \equiv ini_request_i.(assign_identifier_i, 2).ini_response_i. \\
&\qquad purchase_request_i.(m_verify_i, 1).(creat_autho_i, 3). \\
&\qquad autho_request_i.autho_response_i.purchase_response_i. \\
&\qquad capture_request_i.capture_response_i.M_i \\
&pay_gateway_i \equiv P_1 \parallel P_2 \parallel \ldots \parallel P_i \parallel \ldots \parallel P_m \\
&P_i \equiv autho_request_i.(p_verify_i, 1).issuer_autho_req_i. \\
&\qquad (autho_issuer_i, 2).issuer_autho_res_i.autho_response_i. \\
&\qquad capture_request_i.(p_verify_i, 1).pay_capture_req_i. \\
&\qquad (pay_issuer_i, 5).pay_capture_res_i.capture_response_i. \\
&\qquad pay_gateway_i.
\end{aligned}$$

6. CONCLUSIONS AND FUTURE WORK

We have presented a formal specification of the purchase phase of the SET protocol in the language **BTC** that takes into account the number of resources the system has at its disposal. The first approach of this algebra has been extended to allow us to include in the specification the set of actions which need one of the shared resources for their execution and the number of resources available in the system. In this new approach we are able to deal with heterogeneous resources.

With the formal resource-aware model we have defined, the timed characteristics of the system have been captured. Now, starting from the operational semantics, we are able to make a transition graph to help us with the performance evaluation and we can use the algorithm we defined in [19] to estimate the time required to evolve between states. With this transition graph we have studied the SET system and calculated the time required for it to evolve from the initial state to the final one, with several values in the number of cardholder processes in the system and changes in the number of real available merchants [20], but in this paper we worked just with homogeneous resources. As soon as we finish the tool to generate the transition graph from the specification, we will calculate these values in system with heterogeneous resources.

We consider our model suitable to assist in the design of the system because it allows us to find the minimum number of resources needed to obtain the best performance of the system and it could also be useful to evaluate the performance of a system when the number of shared resources is fixed.

We believe the need for a tool to generate the transition graph from the specification is clear and, of course, this is to be one of our future goals.

In this paper we have focused our efforts on the purchase phase of SET however our future work will cover the remaining phases of the protocol.

7. REFERENCES

[1] G. Bella, F. Massacci, and L.C. Paulson. The verification of an industrial payment protocol: The SET purchase phase. In *Proceedings of 9th ACM Conference on Computer and Communications Security*, pages 12–20, 2002.

[2] G. Bella, F. Massacci, and L.C. Paulson. Verifying the SET registration protocols. In *Proceedings of IEEE J. of Selected Areas in Communications*, pages 21(1):77–87, 2003.

[3] P. Brémond-Grégoire and I. Lee. A process algebra of communicating shared resources with dense time and priorities. In *Theoretical Computer Science* ***189***, pages 179–219, 1997.

[4] Mikael Buchholtz, Jacob Andersen, and Hans Henrik Løvengreen. Towards a Process Algebra for Shared Processors. *Electronic Notes in Theoretical Computer Science*, 52, 2002.

[5] G. Costa and C. Stirling. Weak and strong fairness in CCS. *Information and Computation*, 73:207–244, 1987.

[6] F. Cuartero, D. de Frutos, and V. Valero. A sound and complete proof system for probabilistic processes. In *ARTS'96, LNCS 1231*, 1997.

[7] A.O. Freier, P. Karlton, and P.C. Kocher. The SSL protocol v3.0. *Internet Draft*, 1996.

[8] A. Giacalone, C.-C. Jou, and S.A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proceedings of Working Conference on Programming Concepts and Methods, IFIP TC 2*, Sea of Galilee, Israel, 1990.

[9] Damas P. Gruska. Process Algebra for Limited Parallelism. In *Proc. of Concurrency, Specification and Programming (CS&P'96)*, pages 61–74, 1996.

[10] Damas P. Gruska. Bounded Concurrency. In *In Chlebus, B.S., Czaja, L., eds.: Proc. of 11th International Symposium on Fundamentals of Computation Theory (FCT'97), LNCS 1279, Springer*, pages 198–209, 1997.

[11] H. Hermanns, U. Herzog, and J-P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 274(1–2):43–86, 2002.

[12] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[13] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[14] I. Lee, P. Brémond-Grégoire, and R. Gerber. A process algebraic approach to the specification and analysis of resource-bound real-time systems, 1994.

[15] Mastercard and VISA. *SET Secure Electronic Transaction Specification.* http://www.setco.org/set_specifications.html, 1997.

[16] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *CONCUR'90, LNCS 458*, pages 401–415, Amsterdam, 1990.

[17] M. Núñez, D. de Frutos, and L. Llana. Acceptance trees for probabilistic processes. In *CONCUR'95, LNCS 962*, pages 249–263, 1995.

[18] G.M. Reed and A.W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58:249–261, 1988.

[19] M.C. Ruiz, D. Cazorla, F. Cuartero, J.J. Pardo, and H. Macia. A bounded true concurrency process algebra for performance evaluation. In *Proc. of the 1st European Performance Engineering Workshop (EPEW'04), LNCS 3236*, pages 143–155, Toledo, Spain, October 2004. Springer.

[20] M.C. Ruiz, D. Cazorla, F. Cuartero, J.J. Pardo, and H. Macia. A formal specification and performance evaluation of the purchase phase in the SET protocol. In *Proc. of the 7th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (CAVIS'05)*, pages 10–15, Timisoara, Romania, September 2005.

[21] J.J.M.M. Rutten and J.I. Zucker. A semantics approach to fairness. *Fundamenta Informaticae*, XVI:1–38, 1992.

[22] D.H. Steves, C. Edmondson-Yurkanan, and M. Gouda. A protocol for secure transactions. In *Proceedings of the Second USENIX Workshop on Electronic Commerce.*, pages 201–212, 1996.