AN AGENT BASED APPROACH TO MODELING THE SECURE ELECTRONIC TRANSACTION PROTOCOL

by
D.N. Kleftouris, N. Maragos, C. Ziogou, Ch. Mouchos

Abstract: Secure Electronic Transaction (SET) is an open protocol, which has the potential to emerge as a powerful tool in securing of electronic transactions. It is of primary importance to produce formal specifications that describe precisely the functional and temporal properties of the protocol leading to validation and verification prior to committing to implementation. The primary objective of this work is the construction of an agent based model for the operation of the SET protocol. A theoretical framework for the agents is presented and an architectural diagram based on them is constructed. Finally, formal specifications for every agent participating in the model are developed.

1. INTRODUCTION

E-Commerce (Turban E., 2002) is a distributed environment where a number of trading parties such as customers, merchants and service providers, collaborate with the aid of information technologies to conduct business transactions. With the dominance of Internet, electronic commerce moves over the Net and thus new technological and communicational standards and specifications (Ouyang, C., 2002) emerged to provide the required infrastructure. Secure Electronic Transaction is a Protocol aims at providing an interoperable framework for secure electronic business transactions between the participants of an E-Commerce environment, which have no prior association between them. The major effort of the SET protocol is to secure the payments in a consumer-to-business e-commerce environment.

It is of primary importance to develop a sound architectural model of the Protocol to simplify the construction of complex e-commerce configurations on one hand, and serve as a test bed for evaluating suggested new business strategies on the other. Formal modeling and analysis of agent-based architectures promotes understanding and reasoning and can result to models that describe precisely the functional and temporal properties of the system leading to validation and verification prior to undertaking any implementation effort.

The primary objective of this paper is the construction of an agent-based model for the operation of the SET protocol. Initially a theoretical framework for the agents is presented and an architectural diagram based on them is constructed. Finally, state machine chart diagrams and formal specifications for every agent participating in the model are developed and general concluding remarks are done.

2. AGENT BASED SYSTEMS

An E-commerce architecture is a software system which consists of three types of distinct building blocks, agents, connectors and configurations. An agent is a software component, which continuously runs, it exists as a semi-autonomous entity, and performs various activities for the completion of a transaction.

An agent is an object with the following properties (Alagar, V.S., 2001):

- An agent is a software component made to achieve certain objectives.
- An agent can be either autonomous or can respond to stimuli which are other agents.
- An agent has enough computational resources and knowledge in order to complete the tasks assigned to it within certain time limits.
- An agent communicates with other agents in its environment with messages at its ports, where a port is an access point for a bidirectional communication.
- An agent can change dynamically its behavior if its content is changed.

A connector is a channel of communication between two agents and is described by the protocols of message exchanges through that connector (Allen, R., 1999). A configuration defines the cooperation between a finite number of agents with connectors along which they communicate. Each configuration models a role. An agent can participate in various roles.

The global context for an e-commerce system is defined as the tuple of GC (A, R, P) where:

- A is a finite set of agents, where each of theses agents has a finite set of port types
- R=MUm, where M is a set of messages for communications between the agents, m={create, dispatch, engage, disengage, dispose, silent} are control messages and MUm = Ø
- P is a finite set of applications.

A port type defines a set of messages that can occur at a port of this type. Port types are defined with the symbol @. An agent of the type A [L] $\in$ A, where L is the list of the port types, can be created by initializing every port type in L with a finite number of ports and by assigning these ports to the agent. For example $A_1[p_1,p_2 : @P ; q_1,q_2,q_3 : @Q]$ is an agent of the type A[@P,@Q]. Messages of a specific type can only be sent or received by ports of the same type.

All agents have modes. When a new agent is created its ports, id and characteristics are initialized in mode "initial". Initial mode is the result of the control message creates. After its creation the agent is in sate wait in which it waits to act. An agent can be in this state in its home or in a removed site. However it can be send to another site through the message dispatch and then it is in state dispatched. When the agent receives the message engage, agent's mode changes from dispatched to remote-run and it executes its remote task. When it finishes the task the agent goes to mode

pause. If there are no more tasks in that site the agent gets the message disengage and goes to mode wait again. An agent in a remote site can be recalled using the message recall. When it receives this message it turns its mode to retract. Then the message engage changes the agent's mode from retract to local-run, and when the agent finishes its work it goes again to mode pause. When an agent, who is in mode wait, receives the message dispose it changes its mode to dispose.

Structures for the specification of agent types and System Configuration are the following:

Agent <identifier>[<port types>] Events: States: Attributes: Attribute-Function: Transition-Specification: Time-Constraints: End	Subsystem <identifier> Include: Instantiate: Configure: End

The field "Events" describes all the messages that exist it the system. Each event is marked with a "!" or a "?" if it is for input or output accordingly. Field "States" describes all the situations in which the agent can be found according to the events it receives. The field "Transition-Specifications" describes all the states in which the agent can occur. The field "Time-Constraints" defines the time bounds if the agent is timed.

Each agent answers to every message it receives. The message is of the shape <e, p_i, t>, which means the message e occurs at time t at port p_i. Two ports are compatible if the set of input messages at one port is equal to the set of output messages at the other. If the port p of agent Ai is connected to one of each compatible port q of agent A_j then the composition relation $A_i.@p \leftrightarrow A_j.@p$ is valid. When a message is exchanged through the channel connecting two agents then the agents change their statuses in the same time.

3. SET PROTOCOL OVERVIEW

Secure Electronic Transactions (SET) Protocol is based on the science of cryptography and the art of encoding and decoding messages. It relies on two different encryption mechanisms, as well as an authentication mechanism. SET uses symmetric encryption, in the form of the Data Encryption Standard (DES), as well as asymmetric, or public-key, encryption to transmit session keys for DES transactions (Visa, 1997). Rather than offer the security and protection afforded by public-key cryptography, SET simply uses session keys (56 bits), which are transmitted asymmetrically, and the remainder of the transaction uses symmetric encryption in the form of DES. This has

disturbing effects to a "secure" electronic transaction protocol because public key cryptography is only used to encrypt DES keys and for authentication, and not for the encryption of the main body of the transaction. The computational cost of asymmetric encryption is cited as the reason for using weak 56 bit DES (Visa, 1997).

The SET protocol has three principle features:

1. All sensitive information sent between the three parties is encrypted.
2. All three parties are required to authenticate themselves with certificates.
3. The merchant never sees the customer's card number in plaintext.

This last feature actually makes commerce on Internet more secure than traditional credit card transactions, such as credit card transactions over phone. Indeed, if you were to order a sweater over the phone, you would leave your card number with the merchant. Someone working for the merchant could then obtain your card number and make purchases at your expense. But with the SET protocol, only the merchant's bank gets to see the card number in plaintext.

Four different agents are the main constituents in SET protocol, around which the whole philosophy of the protocol is built. These agents are Cardholder, Merchant, Payment Gateway and Certificate Authority. The role that each one plays in the e-commerce process is the following:

- Cardholder, who is an owner of a credit card issued by an Issuer Bank that also provides support for e-commerce transactions.
- Merchant, who is the one that provides goods, services and information and accepts to be paid using the SET protocol.
- Payment Gateway is a system, which offers e-commerce services to merchants with the support of the Acquirer and acts as its representative to undertake the capture and the approval of a transaction.
- Certificate Authority is an agency that represents one or more credit card companies and is responsible for the supply and distribution of authentication certificates towards the Cardholder, the Merchant and the Payment gateway.

In addition to Agents described above the following secondary Entities take part in the process.

- Issuer is a finance organization, which supports the publication of credit cards.
- Acquirer is a finance organization that stakes merchants to accept credit cards for their transactions.
- Finance Network of the credit card’s company is the network that is used by the company to connect the Acquirer and the Issuer.

For the achievement of a transaction the following sequence of actions is followed. The cardholder, using a computer, buys goods in the World Wide Web (www) by choosing an object that is online in a catalogue. When the cardholder decides to buy some goods, the operation of the protocol begins to ensure a secure electronic transaction for the cardholder. The transaction of a purchase can take place in many different ways, depending on the professional situation of the merchant and the way the cardholder wants the transaction to be done. For example:

- The cardholder may want to pay in installments.
- The order may contain goods that should be sent by post or some product that is not in material form, like a video clip, which can be sent by e-mail.

In Fig. 1 the agents in the protocol with their interconnections are presented diagrammatically.

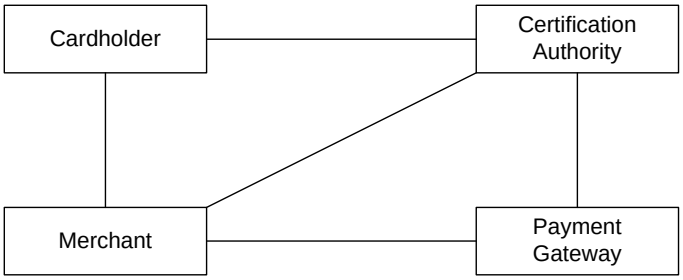

Fig. 1. General Description of SET Protocol

From the general description of the system presented in Fig. 1, taking into account the communication links, communication messages exchanged between the agents, the types of all agents together with the types of ports for each one are determined. For example, agent cardholder needs two types of ports, one for communication with agent Certificate Authority and the other for communicating with agent Merchant. Thus the general description of the system in Fig.1 by adding the information of ports in each agent becomes detailed and more informative. The improved model is the high-level architecture diagram of the system and is shown in Fig. 2.

4. STATE MACHINE MODELS FOR THE AGENTS

The functional and time behavior of an agent is modeled by a state machine extended with ports, hierarchical states, and transitions governed by clocks and guards (Chen, Q., 2000). The running behavior of the agent depends on the context, defined by the set of messages that can be received or sent to other agents in a specific application. Messages are either input or output events. Messages are received and send at the ports. All ports of a specific type can receive or send only those messages associated with that type. Every agent has a finite set of attributes. The static attributes are the resources at its disposal, tables of information and rules for encoding knowledge and functions to perform the tasks that it has to execute. The dynamic attributes are those required for the agent's interaction in different contexts.

In the sequel the state machines for the four agents of the SET protocol are shown in Fig 3a, 3b, 3c, 3d presenting their behavior when their mode of operation is

either **local-run** or **remote-run**. In the state machine chart diagrams events that occur and states are marked.

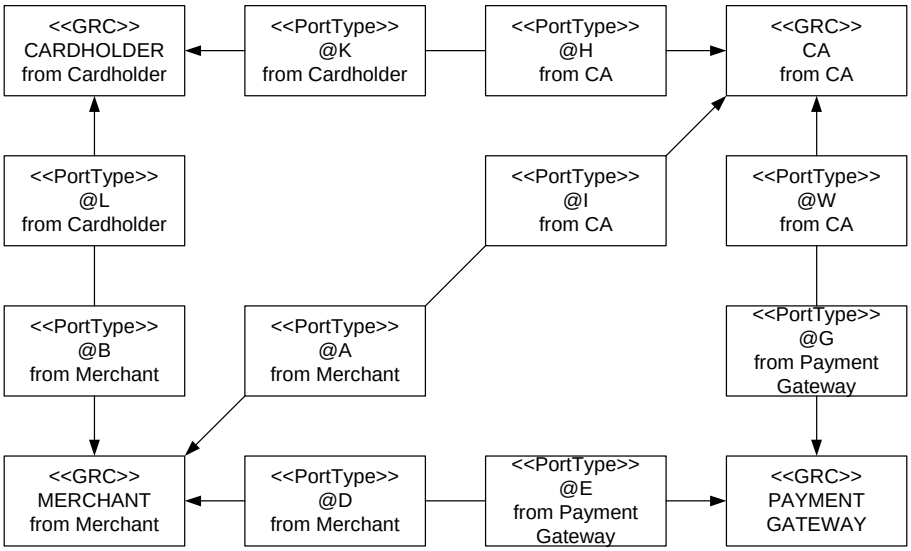

Fig. 2. High-Level Architecture Diagram

5. FORMAL SPECIFICATIONS OF THE AGENTS AND SYSTEM CONFIGURATION

Formal modeling of an E-commerce architecture enables a rigorous analysis of the high level architecture to prove that the desired properties of the specified business scenarios and strategies are verified. Also generic architectures can be considered for modeling and studying similar systems in a broader application domain. In the sequel formal specifications for the System Configuration and for each of the four agents according to formal notation and the operational semantics established to define types of agents and system configurations as described in the 2nd chapter, are cited.

5.1 Formal Specifications for System Configuration

SET Ecommerce
    Includes:
    Instantiate:
        Cl::Cardholder[@K:l, L@:1];
        Ml::Merchant[@B:l, @A:1, @D:1];

```
        Pl::PaymentGateway[@G:l, @E:1];
        CAl::CertificateAuthority[@H:l,@I:1,@W:1];
Configure:
        C1.@K1:@K< -- >CA1.@H1:@H;
        C1.@L1:@L< -- >M1.@B1:@B;
        M1.@A1:@A< -- >CA1.@I1:@I;
        M1.@D1:@D< -- >P1.@E1:@E;
        CA1.@W1:@W< -- >P1.@G1:@G;
End
```

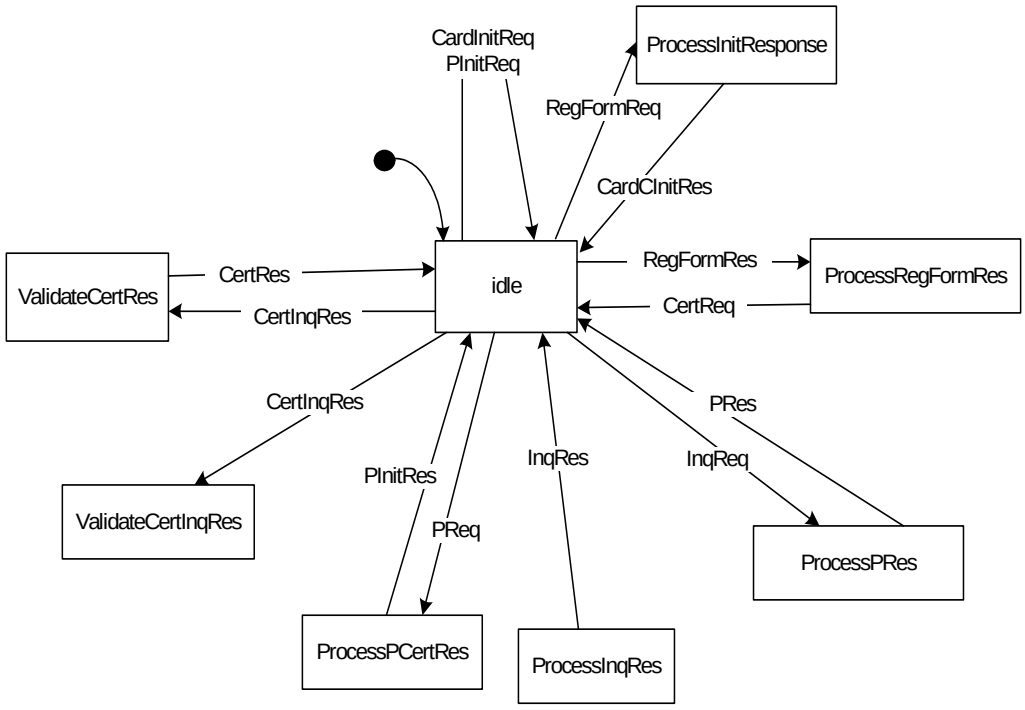

Fig.3a. State Machine Chart Diagram for Cardholder

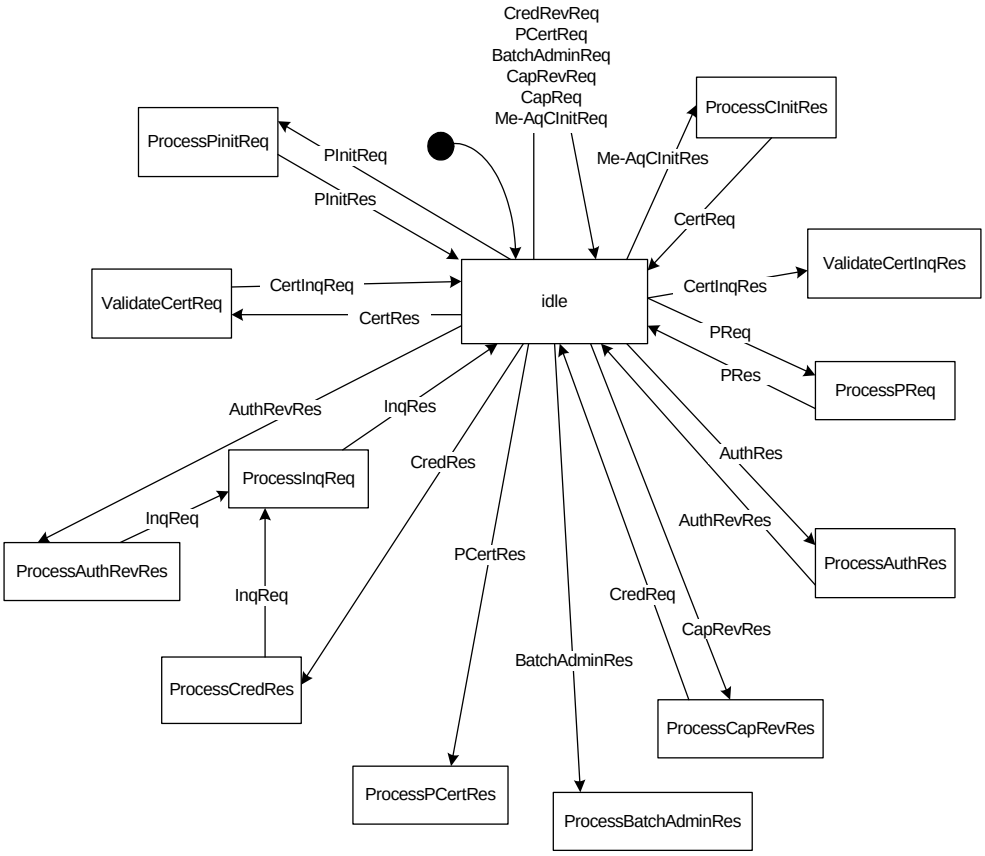

Fig. 3b. State Machine Chart for Merchant

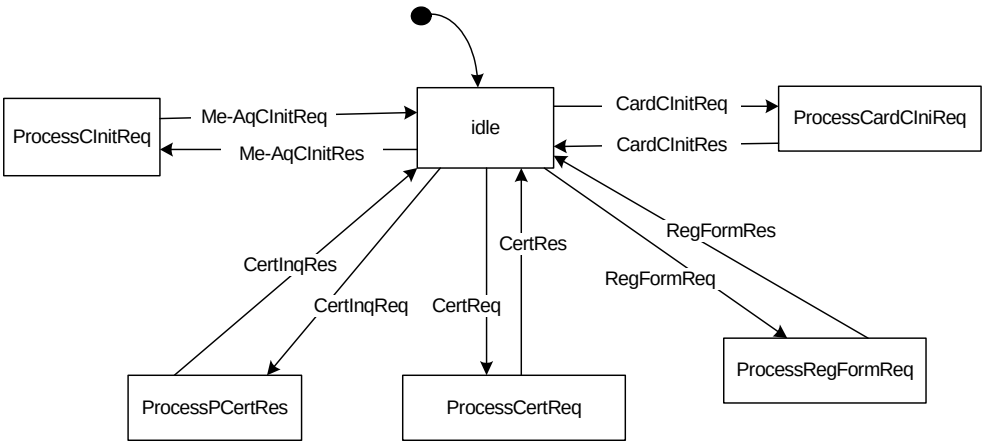

Fig. 3c. State Machine Chart diagram for Certificate authority

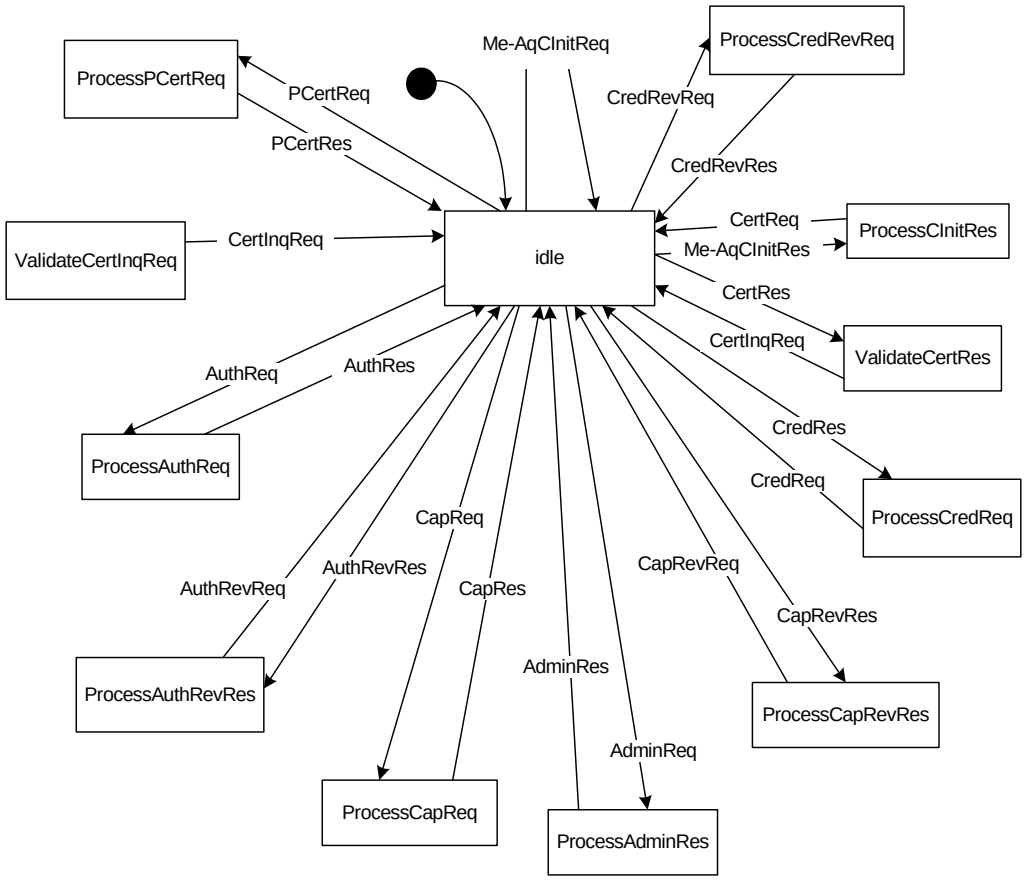

Fig. 3d. State Machine Chart Diagram for Payment Gateway

5.2 Formal Specifications for the Cardholder

Agent Cardholder[@K, @B]
Events: CardCInitReq!@K, CardCInitRes?@K, RegFormReq!@K, RegFormRes?@K, CertReq!@K, CertRes?@K, CertInqReq!@K, CertInqRes?@K, PInitReq!@L, PInitRes?@L, PReq!@L, PRes?@L, InqReq!@L, InqRes?@L
States: *idle, ProcessInitResponse, ProcessRegFormRes, ValidateCertRes, ValidateCertlnqRes, ProcessPInitRes, ProcessInqRes, ProcessPres
Attributes:
Attribute-Function: idle—> {} ;ProcessInitResponse—> {} ;ProcessRegFormRes—> {};
ValidateCertRes-* {}; ValidateCertlnqRes—> {} ;ProcessPInitRes—> {};
ProcessInqRes—> {} ;ProcessPres—>•
{}; Transition-Specifications:
Rl: <idle,idle>;CardCInitReq(true);true => true;

R2: <idle, ProcessInitResponse>; CardCInitRes(true); true => true;
R3: <ProcessInitResponse, idle>; RegFormReq(true); true => true;
R4: <idle, ProcessRegFormRes>; RegFormRes(true); true => true;
R5: <ProcessRegFormRes, idle>; CertReq(true); true => true;
R6: <idle, ValidateCertRes>; CertRes(true); true => true;
R7: <ValidateCertRes, idle>; CertlnqReq(true); true => true;
R8: <idle, ValidateCertInqRes>; CertlnqRes(true); true => true;
R9: <idle, idle>; PlnitReq(true); true => true;
RIO: <idle, ProcessPInitRes>; PlnitRes(true); true => true;
Rll: <ProcessPInitRes, idle>; PReq(true); true => true;
R12: <idle, ProcessPres>; PRes(true); true => true;
R13: <ProcessPres, idle>; InqReq(true); true => true;
R14: <idle, ProcessInqRes>; InqRes(true); true => true;
Time-Constraints: End

5.3 Formal Specifications for the Merchant

Agent Merchant[@B, @A, @D]
Events: Me-AqCInitReq!@A, Me-AqCInitRes?@A, CertReq!@A, CertRes?@A, CertInqReq!@A, CertInqRes?@A, PInitReq?@B, PInitRes!@B, PReq?@B, PRes!@B, AuthReq!@D, AuthRes?@D, AuthRevReq!@D, AuthRevRes?@D, InqReq?@B, InqRes!@B, CapReq!@D, CapRes?@D, CapRevReq!@D, CapRevRes?@D, CredReq!@D, CredRes?@D, CredRevReq!@D, CredRevRes?@D, PCertReq!@D, PCertRes?@D, BatchAdminReq!@D, BatchAdminRes?@D
States: *idle, ProcessCInitRes, ValidateCertRes, ValidateCertlnqRes, ProcessPInitReq, ProcessPReq, ProcessAuthRes, ProcessAuthRevRes, ProcessInqReq, ProcessCapRes, ProcessInqReq, ProcessCapRevRes, ProcessCredRes, ProcessPCertRes, ProcessBatchAdminRes
Attributes:
Traits: idle—»{};
Attribute-Function: idle—>•{}; ProcessCInitRes—>{}; ValidateCertRes—*{}; ValidateCertlnqRes^ {}; ProcessPInitReq—> {} ;ProcessPReq, ProcessAuthRes^- {}; ProcessAuthRevRes—»• {}; ProcessInqReq—>• { }; ProcessCapRes, ProcessInqReq—*{}; ProcessCapRevRes^{}; ProcessCredRes^> { } ;ProcessPCertRes—»• { }; ProcessBatchAdminRes—* { };
Transition-Specifications:
R1: <idle, idle>; Me-AqCInitReq(true); true => true;
R2: <idle, ProcessCInitRes>; Me-AqCInitRes(true); true ^> true;

R3: < ProcessCInitRes, idle>; CertReq(true); true => true;
R4: <idle, ValidateCertRes>; CertRes(true); true ^> true;
R5: < ValidateCertRes, idle>; CertlnqReq(true); true => true;
R6: < idle, ValidateCertInqRes>; CertlnqRes(true); true => true;
R7: <idle, ProcessPInitReq>; PlnitReq(true); true => true;
R8: <ProcessPInitReq, idle>; PlnitRes(true); true => true;
R9: <idle, ProcessPReq>; PReq(true); true => true;
RIO: <ProcessPReq, idle>; PRes(true); true => true;
Rll: <idle, idle>; AuthReq(true); true => true;
R12: <idle, ProcessAuthRes>; AuthRes(true); true => true;
R13: <ProcessAuthRes, idle>; AuthRevReq(true); true => true;
R14: <idle, ProcessAuthRevRes>; AuthRevRes(true); true =^> true;
R15: <ProcessAuthRevRes, ProcessInqReq>; InqReq(true); true => true;
R16: <ProcessInqReq, idle>; InqRes(true); true => true;
R17: <idle, idle>; CapReq(true); true => true;
R18: <idle, processCapRes>; CapRes(true); true => true;
R19: <ProcessCapRes, ProcessInqReq>; InqReq(true); true => true;
R20: <ProcessInqReq, idle>; InqRes(true); true => true;
R21: <idle, idle>; CapRevReq(true); true => true;
R22: <idle, ProcessCapRevRes>; CapRevRes(true); true => true;
R23: <ProcessCapRevRes, idle>; CredReq(true); true => true;
R24: <idle, ProcessCredRes>; CredRes(true); true => true;
R25: <ProcessCredRes, ProcessInqReq>; InqReq(true); true => true;
R26: <ProcessInqReq, idle>; InqRes(true); true => true;
R27: <idle, idle>; CredRevReq(true); true => true;
R28: <idle, ProcessCredRes>; CredRevRes(true); true => true;
R29: <idle, idle>; PCertReq(true); true => true;
R30: <idle, ProcessPCertRes>; PCertRes(true); true => true;
R31: <idle, idle>; BatchAdminReq(true); true => true;
R32: <idle, ProcessBatchAdminRes>; BatchAdminRes(true); true => true;
Time-Constraints:
End

5.3 Formal Specifications for the Payment Gateway

Agent Payment_Gateway [@G, @E]
Events: Me-AqCInitReq!@G, Me-AqCInitRes?@G, CertReq!@G, CertRes?@G,
CertInqReq!@G, CertInqRes?@G, AuthReq?@E, AuthRes!@E,

AuthRevReq?@E, AuthRevRes!@E, CapReq?@E, CapRes!@E,
CapRevReq?@E, CapRevRes!@E, CredReq?@E, CredRes!@E,
CredRevReq?@E, CredRevRes!@E, PCertReq?@D, PCertRes!@E,
BatchAdminReq?@E, BatchAdminRes! @E
States: *idle, ProcessCInitRes, ValidateCertRes, ValidateCertlnqRes,
ProcessAuthReq, ProcessAuthRevReq, ProcessCapReq,
ProcessCapRevReq, ProcessCredReq, ProcessPCertReq, ProcessAdminReq
Attributes:
Traits:
Attribute-Function: idle—>{}; ProcessCInitRes—>{}; ValidateCertRes—»• {};
ValidateCertlnqRes—*!}; ProcessAuthReq—>{}; ProcessAuthRevReq^'{};
ProcessCapReq—>{}; ProcessCapRevReq—*{}; ProcessCredReq—»• {};
ProcessPCertReq—> *{ }*; ProcessAdminReq-^ { };
Transition-Specifications:
Rl:<idle, idle>; Me-AqCInitReq(true); true => true;
R2: <idle, ProcessCInitRes>; Me-AqCInitRes(true); true => true;
R3: <ProcessCInitRes, idle>; CertReq(true); true => true;
R4: <idle, ValidateCertRes>; CertRes(true); true => true;
R5: <ValidateCertRes, idle>; CertlnqReq(true); true => true;
R6: <idle, ValidateCertInqRes>; CertlnqRes(true); true => true;
R7: <idle, ProcessAuthReq>; AuthReq(true); true => true;
R8: <ProcessAuthReq, idle>; AuthRes(true); true ^> true;
R9: <idle, ProcessAuthRevReq>; AuthRevReq(true); true => true;
RIO: <ProcessAuthRevReq, idle>; AuthRevRes(true); true =^> true;
Rll: <idle, ProcessCapReq>; CapReq(true); true => true;
R12: <ProcessCapReq, idle>; CapRes(true); true => true;
R13: <idle, ProcessCapRevReq>; CapRevReq(true); true => true;
R14: <ProcessCapRevReq, idle>; CapRevRes(true); true ^> true;
R15: <idle, ProcessCredReq>; CredReq(true); true => true;
R16: <ProcessCredReq, idle>; CredRes(true); true => true;
R17: <idle, ProcessCredRevReq>; CredRevReq(true); true => true;
R18: <ProcessCredRevReq, idle>; CredRevRes(true); true => true;
R19: <idle, ProcessPCertReq>; PCertReq(true); true => true;
R20: <ProcessPCertReq, idle>; PCertRes(true); true =i> true;
R21: <idle, ProcessAdminReq>; BatchAdminReq(true); true =^> true;
R22: <ProcessAdminReq, idle>; BatchAdminRes(true); true => true;
Time-Constraints:
End

5.5 Formal Specifications of the Certificate Authority

Agent CertificateAuthority[@H, @I, @W]
Events: CardCInitReq?@H, CardCInitRes?@H, RegFormReq!@H,
RegFormRes?@H, CertReq!@H, CertRes?@H, CertInqReq!@H,
CertInqRes?@H, Me-AqCInitReq! @I, Me-AqCInitRes?@I,
CertReq!@I, CertRes?@I, CertInqReq!@I, CertInqRes?@I, Me-AqCInitReq!@W,
Me-AqCInitRes?@W, CertReq!@W, CertRes?@W, CertInqReq!@W,
CertInqRes?@W
States: *idle, ProcessCardCInitReq, ProcessRegFormReq, ProcessCertReq,
ProcessCertlnqReq,
ProcessCInitReq Attributes:
Traits:
Attribute-Function: idle—»•{}; ProcessCardCInitReq—^{}; ProcessRegFormReq—»{};
ProcessCertReq—*{}; ProcessCertlnqReq—>{}; ProcessCInitReq-^{};
Transition- Specifications:
Rl: <idle, ProcessCardCInitReq>; CardCInitReq
R2: <ProcessCardCInitReq, idle>; CardCInitRes
R3: <idle, ProcessRegFormReq>; RegFormReq
R4: <ProcessRegFormReq, idle>; RegForniRes
R5: <idle, ProcessCertReq>; CertReq
R6: <ProcessCertReq, idle>; CertRes
R7: <idle, ProcessCertInqReq>; CertlnqReq
R8: <ProcessCertInqReq, idle>; CertlnqRes R9: <idle, ProcessCInitReq>;
Me-AqCInitReq RIO: <ProcessCInitReq, idle>; Me-AqCInitRes Rl 1: <idle,
ProcessCertReq>; CertReq R12: <ProcessCertReq, idle>; CertRes R13:
<idle, ProcessCertInqReq>; CertlnqReq R14: <ProcessCertInqReq, idle>;
CertlnqRes R15: <idle, ProcessCInitReq>; Me-AqCInitReq R16:
<ProcessCInitReq, idle>; Me-AqCInitRes R17: <idle, ProcessCertReq>;
CertReq R18: <ProcessCertReq, idle>; CertRes R19: <idle,
ProcessCertInqReq>; CertlnqReq R20: <ProcessCertInqReq, idle>;
CertlnqRes
Time-Constraints:
End

6. CONCLUSIONS

An agent based model, for the E-Commerce Protocol SET has been developed. Initially, the High Architectural Diagram of the system was designed, where the agent types with their ports and interconnections for message exchanges

were defined. State machine chart diagrams modeling the operation of every agent have been constructed. Furthermore, formal specifications have been produced to describe analytically, in a mathematical form the specification of every agent type and of the system configuration. The implementation of agents strictly conforms to their state machine descriptions.

Agent based software execution modeling offers understanding and reasoning of the functional and temporal specifications of the software implementation prior to undertaking any development. This way validation and verification of a new software product can be performed in a concise and systematic manner. Simulated executions of the formal model can be conducted to study and investigate many of the issues raised in (Griss, 2000) such as Effectiveness (For a given set of customers what business strategy is more effective), Stability (Should customer characteristics change a bit which business strategy is least affected), Timeliness (Do customers are served by the system in a reasonable time in a realistic environment with many calls ?).

REFERENCES

1. Alagar, V.S. and Zheng Xi (2001). *A Rigorous Approach to Modeling and Analyzing E-Commerce Architectures.* LNCS 2021, Springer Verlang, FME 2001, pp 173-196.
2. Allen, R. and D. Muthiayen (1999). *A Formal Basis for Architectural Connection*. ACM Transactions on Software Engineering and Methodology.
3. Chen, Q., M. Hsu, U. Dayal and M. Griss (2000). *Multi-Agent Cooperation, Dynamic Workflow and XML for E-Commerce Automation.* In Proceedings Autonomous Agents 2000, Barcelona, Spain, June 2000.
4. Griss, M.L. and R. Letsinger (2000). *Games at Work : Agent-Mediated E-Commerce Simulation*. In Proceedings Autonomous Agents 2000, Barcelona, Spain, June 2000.
5. Ouyang, C., L.M. Kristensen and J. Billington (2002). *A Formal and Executable Specification of the Internet Open Trading Protocol.* LNCS 2455, Springer Verlang, pp. 377-387.
6. Visa and MasterCard (1997). *SET Secure Electronic Transaction Specification*. Vols. 1,2,3, Version 1.0.
7. Turban E., et all. (2002). *Electronic Commerce: A Managerial Perspective*. Prentice Hall.

Authors :

D.N. Kleftouris,
N. Maragos, C. Ziogou, Ch. Mouchos
Dept of Information Technology, Technological Educational Institute of Thessaloniki, Thessaloniki 546 06, Greece, Email : { nmarag, klefturi, ziochr }@it.teithe.gr.