Context-Aware Payment for Supply Chains: Software Architecture and Formal Verification

Zahra Zamani, Maryam Bayat, Ali Moeini, Alireza Motevalian

***Abstract*— Lack of automation in supply chain payment systems is one of the missing points in the area of optimizations and can cause inefficiencies in the total performance of the chain. To overcome this problem, a new context-aware payment method is introduced in this paper, based on pervasive and ubiquitous computing technologies. The main architectural structure is inspired from the Service-Oriented Context Aware Middleware (SOCAM) along with the modification of existing architectures in context-aware systems. It also provides components to insure trusted interactions among members in a supply chain. A new three-layered architecture has been suggested to support this new approach. Software architecture of the proposed payment system has been specified using the Unified Modeling Language (UML) and its crucial properties making the system safe and reliable have been verified using *formal methods*. The verification of such approach using Temporal Logic insures its progress and overall performance of the future systems developed under this architecture.**

***Index Terms* — Context-awareness, Formal Methods, Software Architecture, Supply Chain Management.**

I. Introduction

Ubiquitous computing introduces a new method of computing after mainframes and distributed systems. It is often known as the calmest technology for establishing communication between humans (or other objects) with computer systems, as it runs in the background of everyday life of people and tries not to be sensed by human [1].

Currently, the most focused issue in ubiquitous/pervasive computing is Context-awareness. A context-aware system is able in adapting its operations to a given context, without explicit user intervention and thus aims at increasing usability and effectiveness by taking environmental context into account [2]. As proposed by Dey and Abowd, "Context" is defined as *any information that can be used to characterize the situation of entities (i.e., whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves* [2]. According to this definition, three distinct entities are usually identified concerning the context concept, namely Places, People and Things. Each of these entities can be characterized with attributes such as identity, location, status and time.

Recent advances in ubiquitous/pervasive computing, have led to the emergence of a new concept called Ubiquitous Payment defined as ubiquitous, invisible and unobtrusive payment, which is integrated into the environment and regards the context of the payer. This means that the payment process should neither interrupt the payer in his current action nor should not interrupt running processes [3].

On the other hand, automation and optimization of Supply Chain (SC) activities is a newly growing research area which has been focused by many researchers recently. A Supply chain is a coordinated system of organizations, people, activities, information and resources being involved in moving a product or service in physical or virtual manner from the supplier to the customer. Supply chain activities, therefore, transform raw materials and components into a matured product that is delivered to the end customer [4].

Recently, efforts have been conducted to optimize the supply chain activities and minimize the required time and cost for producing a product/service in order to improve the efficiency of the whole chain. However, a missing process in these efforts, which has not been given its proper importance, is the Payment process between the members of a supply chain. As discussed in [5], a great majority of supply chains' members are currently using paper-based payment systems to perform their financial flows (including activities such as initiating, tracking and reconciling paper-based invoices), imposing additional cost to the supply chain as well as causing serious problems in handling tremendous amounts of financial transactions between the members. The potential problems, caused by using paper-based payment systems in companies within a supply chain are identified as follows [5]:

- Difficulties in efficiently tracking the accounts payable (A/P) and receivable (A/R).
- Imposing extra cost for manually tracking detailed information on stock coding (keeping numbers) and items' quantity.
- Monitoring and enforcing conformance to corporate spending policies are almost impossible, since most of expending are done in an ad hoc manner
- Payment is delayed due to the invoice reconciliations.

Such problems urge the necessity of a payment optimization process amongst supply chain members in order to lead to a radical reduction in cost and time as to improve the overall efficiency of the supply chain. This paper attempts

Manuscript received March 17, 2008.

Zahra Zamani is with the Algorithm and Computations Department, Faculty of Engineering, University of Tehran, Tehran, Iran (phone:0098-21-88308307; fax: 0098-21- ; email: zahrazamani@ut.ac.ir).

Maryam Bayat. is with the Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran (e-mail: maryam.bayat@aut.ac.ir).

Ali Moeini. is Associate Professor with the Algorithm and Computations Department, Faculty of Engineering, University of Tehran, Tehran, Iran (email: moeini@ut.ac.ir).

Alireza Motevalian is with the Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran (e-mail: motevalian@ce.sharif.edu).

to highlight a new payment process based on ubiquitous payment and context-awareness concepts in order to avoid the mentioned problems in supply chain management. Some of the architectural viewpoints of the proposed process have been specified based on Garland and Anthony's [11] approach using Unified Modeling Language (UML). To prove safety of the proposed architecture and more importantly, to shows that systems implemented based on it progress in an efficient way, we adopted a formal verification technique. The main properties of the system are verified using the Temporal Logic verification language and a set of automatic verifying tools.

The main sections throughout this work are as follows: Section 2 introduces the architecture of the proposed payment system that is to be used beside a Supply Chain Management (SCM) software; section 3 proceeds with the specifications of this payment system using UML as the specification language; verification of some important properties of the proposed model (as a reactive model) is presented in section 4; finally, section 5 draws some concluding remarks.

II. Context-Aware Payment for Supply Chains (CAPSC)

Recent architectural updates on context-aware systems, applying special solutions to realize the concept of context-awareness, include the following sections:

a- Context Managing Framework; a hierarchical framework which utilizes a layered architecture [6].
b- The layered architecture used in the Hydrogen project [7].
c- Context Broker Architecture (CoBrA) the model which uses agents to make context-aware computations in smart spaces possible [8].
d- The peer-to-peer architecture for context-aware computing, named Context Toolkit [9].
e- Service-oriented Context-aware Middleware(SOCAM), the proposed architecture for prototyping and rapid construction of mobile context-aware services [10].

The last item is a distributed middleware that utilizes two-level ontology for defining and storing context data received from the environment in a machine-readable form. SOCAM arranges its components in a three-layered architecture, a layer for context providers, one for context interpreter and databases and the last layer for providing services. Based on a logical interpretation of context information, such middleware provides different services using specific information about the environmental context to make right decisions. Thus, the services utilizing SOCAM as a mean to be aware about the environmental context are usually called Context-aware Services.

Concerning the problem of building a context-aware payment system for supply chains, the main architectural structure of Service-Oriented Context Aware Middleware (SOCAM) is used and modified to serve our special purpose. We have employed the 3-layered architecture of SOCAM in our method, adopted to meet our requirements. The main idea of a context-aware payment for supply chains is illustrated in Figure 1. In this figure the producer can participate as a SC Member, requiring raw material from other SC members (Suppliers) in order to produce its products/services. It is assumed that the SCM software is governing the whole chain by not only creating top-level strategies but also informing its members about the unified plans. As a result the SCM software coordinates the demands of producers with the suppliers' products. In our new model, we assume that ubiquitous technologies such as Radio-Frequency Identification (RFID) Tags (capable of storing information) are available and are used by all members of SC. Therefore, the identification and payment information are stored on raw materials, before being sent to the producer. At the producer's side, this information is read and sent automatically to the Context-Aware Payment system. Authentication of the received context based on Public Key Infrastructure (PKI) is an important issue in the proposed context-aware payment method that ensures authenticity of the sender. (, i.e. legality of a member) Also, the payment information received from the environment helps a SC trust this method for accomplishing the desired payment process in a ubiquitous, context-aware manner. After the authentication phase, Context-Aware system performs payments between the financial institutes of suppliers and producers, based on context information received from the environment.

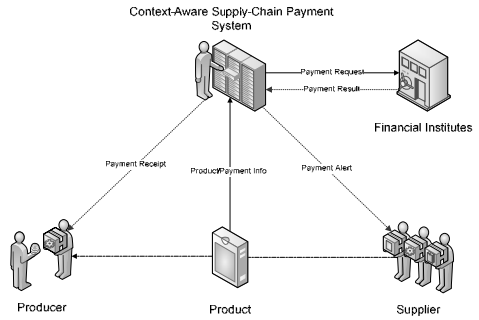

Figure 1: Context-Aware Payment Concept for Supply Chains

Figure 2 shows the proposed architecture of Context-Aware Payment for Supply Chains (CAPSC), which incorporates SOCAM into the above mentioned model. There are changes in each layer of SOCAM based on the new approach. Unlike SOCAM architecture which utilizes services, CAPSC architecture is based on agents in the top layer for accomplishing payment business process along a supply chain. The providers in the bottom layer are defined more explicitly according to SC requirements and the middle layer handles not only interpretation and storage, but also authentication on the context. This three layered architecture consists of Context-Sensing Layer, Perceptual Layer and Payment Agent Layer, a concise description of them follows.

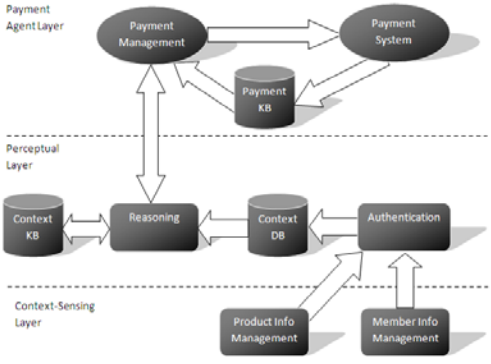

Figure 2: Context-Aware Payment for Supply Chains (CAPSC) Architecture

A. *Context Sensing Layer*

Components of this layer are responsible for sending and receiving context information and are categorized as bellow:

a) Member Info Management Component (MIM): All members in a SC must have trust on Certificate Authority (CA) of the chain and hold a valid certificate from CA either at the initial joining stage or whenever an existing certificate expires. Members then introduce themselves (i.e. its digitally signed identity information) to CAPSC using this component. This information is sent to the Authentication Component for authentication. The MIM should provide a Universal Description Discovery and Integration (UDDI)-like service helping members to locate the provided web services and transferring all members' information to the Perceptual Layer for authentication.

b) Product Info Management Component (PIM): Provides communication interfaces with different context receivers. For example, if RFID Readers are used as context receivers in the supply chain, then this component should provide the CAPSC system with software interfaces which are capable of reading data from different types of used RFID Tags. Indeed, the PIM component knows the raw data arrangement in a tag and retrieves different fields of data carried by products in order to send them back to the Perceptual Layer for authentication and storage.

B. *Perceptual Layer*

Authentication of raw context, interpretation of authenticated data and sending notifications of special events to related agents are performed by this layer which consists of the following components:

a) Authentication Component: Communicates with the SC's trusted CA and authenticates product/member context information by processing the included digital signature. If the received information is distinguished as authenticated, then it will be formatted as a Domain-Specific ontology using Web Ontology Language (OWL) and stored in the Context Database; otherwise it is discarded.

b) Reasoning Component: Reasoning and interpreting new information from raw context is the main responsibility of this component. To achieve this, a Context Database component, containing the last raw context about product/member, and a Context Knowledge-base component, containing results of previous interpretations and historical information about products, is provided in this layer. As an example of a possible interpretation that could be done by this component, one can mention the control of debts of a particular member. When the debt reaches a predefined threshold, the Reasoning Component notifies appropriate agents to pay debts to proper members.

c) Context Database Component: Stores raw member/product context information based on a Domain-Specific ontology.

d) Context Knowledge-base Component: Maintains previously interpreted context and historical product context information.

C. *Payment Agent Layer*

It is supposed that instead of context-aware services in SOCAM architecture, agents perform payments and monitor/manage these transactions in CAPSC. Two general categories of agents in the field of supply chain payment, with the other component in this layer, are explained bellow:

a) Payment Management Agent Component: Manages and monitors payment transactions. For example, receiving debt notifications from the Perceptual Layer and initiating the payment process to proper creditors, setting debt thresholds, creating suitable reports for management and so forth.

b) Payment Agent Component: This type of agent performs the actual payment process between financial institutes of supply chain members. This class of components is treated as external components providing suitable interfaces for its initiators (i.e. Payment Management Agents). As an example, one can name a component performing payment using the Bank Internet Payment System (BIPS) standard via internet.

c) Payment Knowledge-base Component: This component stores historical data about payment transactions performed in the system for further requirements such as reporting.

III. CAPSC Software Architecture Specification

Various approaches may be taken to adequately specify the architecture of systems, such as IEEE1471, Garland and Anthony, Philippe Kruchen's4+1and others [11]. In addition to architectural specification approaches, one can choose an Architectural Definition Language (ADL) or a visual modeling language such as UML as a tool for representing various views of the selected specification approach. ADLs have the advantage of logic and mathematic concepts to specify different views of software architecture. It is noteworthy that, visual specification tools let us easily specify a software architecture using visual modeling.

By adapting Garland and Anthony's approach in specifying large-scale software architectures such as CAPSC, and since UML is commonly used as a specification language due to the fact that understanding visual models is much easier than formal statements, UML is chosen as an effective architectural specification language in this project.

Based on the mentioned approach, three different viewpoints of CAPSC architecture has been developed using UML, namely Conceptual and Analysis viewpoints, Logical Design viewpoints and Environmental viewpoints. Conceptual and Analysis viewpoints are a set of highly abstracted software descriptions, focused on modeling the problem rather than the solution. As an example of the developed models in this viewpoint, Figure 3 is included bellow.

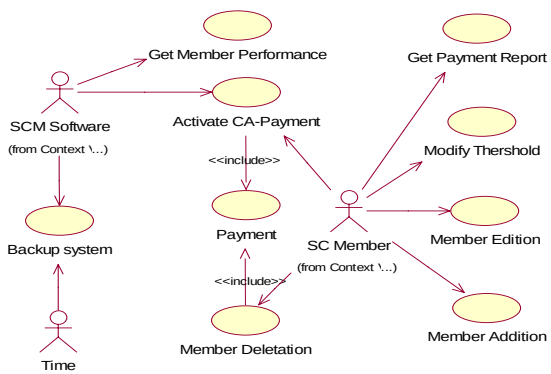

Figure 3: CAPSC Context View model from Conceptual and analysis viewpoint

Logical Design viewpoints are a set of viewpoints targeted at describing the software design. Figure 4 shows an instance of the model.

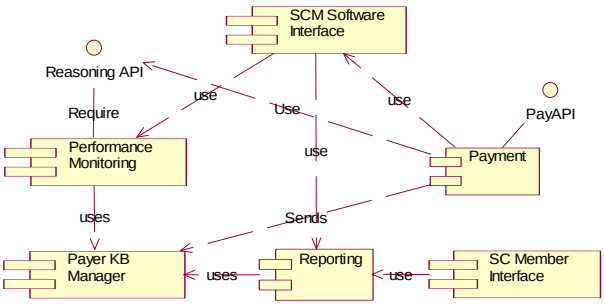

Figure 4: Payment management agent, Component diagram from Logical Viewpoint.

Environmental/Physical viewpoints, focuses on the environment and physical aspects of the software, such as database deployment, that can impact architectural qualities of the system. For abstraction we included the full design of this architecture in [12].

IV. CAPSC Software Architecture Verification

Formal methods are the term used to describe the specification and verification of software and hardware systems using mathematical logic. Formal methods are used to give a description of the system to be developed, at any level(s) of detail desired. This formal description can be used to guide further development activities. It is also used to verify that the requirements for the developing system have been completely and accurately specified [13].

One of the complex domains in software systems are reactive systems. The main role of these systems is to maintain an interaction with their environment, and they must therefore be described in terms of their on-going behavior. Every concurrent and reactive system must be studied by behavioral means. [14]

According to the definitions on reactive systems, we considered CAPSC as a reactive system. Manna and Pnueli in [15] recognized that reactive systems are of growing interest and the Temporal Logic language is well-suited for their formal verification. Usually, Temporal Logic can be classified as the so-called linear-time logic which considers behaviors modeled as a linear sequence of states.

After a system has been modeled, it is useful to provide formal tools to check the validity of properties of the system under specification. Temporal logics have been widely recognized as a useful formalism to express liveness (something good eventually happens) and safety (nothing bad can happen) properties of complex systems [16].

Verification techniques differ according to the architecture being described. The application of various methods such as partial-order reduction, symbolic model checking, symmetric-based, model checking, bit-hash, and compositional verification to UML has been a vast field of study. These methods are described and compared in [17] to come up with a better result for compositional verification in large-scale systems. The CAPSC architecture therefore will have to use this kind of verification when joined with other components of a supply chain. Meanwhile most of the verification done on UML uses the model-checking method because of its ease and speed in checking whether a property holds for a system or not. (See [18], [19] for example)

UML is a semi-formal language, since its syntax and static semantics are defined precisely, but its dynamic semantics are not specified formally [20]. UML consists of different diagrams each serving a special need. State diagrams are used for describing dynamic aspects of a system behavior and since behaviors are greatly considered in verification techniques, therefore our proofs will mainly be focused on these diagrams.

In this paper we have used Temporal Logic as the language for specifying CAPSC properties. The systems behavior is then to be verified automatically using tools developed for this purpose. Various tools are available for the verification of reactive systems with Linear Temporal Logic (LTL) properties such as STeP [21], TABU [18], SMV [22], UPPAAL [23] and SPIN [24]. Most of the tools, appearing in this category, use model-checking as their base for verification. Among these tools, we chose SPIN because of its wide use and ability to verify properties using UML.

As described in [25] Hugo/RT is a UML model translator for model checking theorem proving and code generation: A UML model containing active classes with state machines, collaborations, interactions, and OCL constraints can be translated by Hugo, into the system languages of the real-time model checker UPPAAL, the on-the-fly model checker SPIN, the system language of the theorem prover KIV and into Java and SystemC code. The input to HUGO is an XMI file describing our CAPSC architecture. Among the different components in HUGO, each concentrating on a particular task, we will mainly need this tool to check the consistency of our architecture along with deadlock checks. The output of HUGO is also greatly helpful since it is in the PROMELA form, the input to SPIN, our model checker for verifying the systems' properties.

SPIN is one of the most advanced analysis and verification tools available nowadays. An automatic translation from UML State chart Diagrams to PROMELA (i.e. what we used, Hugo) allows the UML model designer to automatically verify correctness properties of UML State chart Diagram specifications.[24] The tool checks the logical consistency of a specification. It reports on deadlocks, unspecified receptions, flags incompleteness, race conditions, and unwarranted assumptions about the relative speeds of processes. Spin can be used as a full LTL model checking system, supporting all correctness requirements expressible in linear time Temporal Logic, but it can also be used as an efficient on-the-fly verifier for more basic safety and liveness properties.

In order to verify the properties of CAPSC, a short description of the two main categories of Temporal Logic properties is given bellow:

1- **Safety [26] :** Property C is a safety property if the following condition is satisfied for all t ε Σ :
if $\forall i \gg 0, \exists u \in \Sigma \; such \; that \; t(i) \in C \; then \; t \in C$

2- **Liveness [26]:** Property C is a liveness property if and only if $\{t(i): t \in C\} \; is \; \Sigma$. This property is composed of five properties namely Guarantee, Obligation, Response, Persistence and Reactivity.

Before explaining the properties, some global variables and definitions are represented:

- ProductInfo: Used in product management.

- PaymentInfo: The actual payment to be performed by payment agent.
- PaymentManagementInfo: Payment management (PM) activities performed by PM agent.
- KBrequest: Performs storage/retrieval on historical data.
- DBrequest: Performs storage/retrieval on DB according to the request type.
- PKBrequest: Used for payment knowledge-base.
- Performance Request: A management request used to monitor performance.
- ContextNotify: New context generation notification.
- MemberInfo: Add/edit/delete member information requests.
- Init: To initialize each component in CAPSC.
- Shutdown: To unload any component in CAPSC.
- Error: Error handling occurred in any component.

All these global variables are initially set to 0.

$initialize: \forall i \in component |\ init(i) = 1$

$unload: \forall i \in component |\ init(i) = 0 \vee (initialize\ (i) \rightarrow (shutdown = 1))$

We have expressed the properties needed for our system for each component in the definitions bellow:

a) Member Info Management Component:

- Safety Property: It is always the case that the system is waiting to receive a change in member management information.

$\square\ ((initialize\ )\ \mathcal{W}\ (memberInfo\ = 1))$

- Liveness Property: Eventually the component will progress in all stages of its lifecycle.

$\Diamond\ (\ initialize = 0 \vee shutdown = 1)\ \wedge$
$\Diamond\ (initialize = 1) \rightarrow \Diamond\ (memberInfo = 1\ )\ \wedge$
$\square \Diamond \left(\begin{matrix} initialize\ \mathcal{W} \\ (memberInfo = 1) \end{matrix} \right) \vee \neg shutdown$

b) Product Info Management Component:

- Safety Property: It is always the case that the system is waiting to receive new product information.

$\square\ ((initialize\ )\ \mathcal{W}\ (productInfo\ = 1))$

- Liveness Property:

$\Diamond\ (initialize = 0 \vee shutdown = 1)\ \wedge$
$\Diamond\ (initialize = 1) \rightarrow \Diamond (productInfo = 1\ )\ \wedge$
$\square \Diamond \left(\begin{matrix} initialize\ \mathcal{W} \\ (productInfo = 1) \end{matrix} \right) \vee \neg shutdown$

c) Authentication Component:

- Safety Property: It is always the case that the system is waiting to receive product or member information from context-sensing layer components.

$\square (initialize\ )\ \mathcal{W}\ ((productInfo = 1) \vee (memberInfo = 1))$

- Liveness Property:

$\Diamond\ (initialize = 0 \vee shutdown = 1)\ \wedge$
$\Diamond\ (initialize = 1) \rightarrow$
$\Diamond \big((productInfo\ = 1) \vee (memberInfo = 1)\ \big)\ \wedge$
$\square \Diamond (initialize\ \mathcal{W}\ (productInfo\ = 1) \vee (memberInfo = 1)) \vee \neg shutdown$

d) Interpreter Component:

- Safety Property: It is always the case that the system is waiting to receive a notification from other components to perform interpretation using new authenticated context. In our case a notification for performance calculation from the agent layer or context generation due to an update in context database is performed.

$\square ((initialize\ )\ \mathcal{W}$
$\big((PerformanceRequest = 1) \vee (contextNotify = 1)\big) \vee$
$\big((PerformanceRequest = 1) \wedge (error = 1)\big) \vee$
$((contextNotify = 1) \wedge (error = 1)))$

- Liveness Property:

$\Diamond\ (initialize = 0 \vee shutdown = 1)\ \wedge$
$\Diamond (initialize = 1) \rightarrow \Diamond \big((PerformanceRequest = 1) \vee (contextNotify = 1) \big)\ \wedge$
$\Diamond\ \big(\big((PerformanceRequest = 1) \vee (contextNotify = 1) \big) \rightarrow (error = 1) \big) \wedge$
$\Diamond\ ((error = 0\ ) \rightarrow$
$\Diamond \big((PerformanceRequest = 1) \vee (contextNotify = 1) \big) \big) \wedge$
$\square \Diamond \Big(initialize\ \mathcal{W}\ \big((PerformanceRequest = 1) \vee (contextNotify = 1) \big) \Big) \vee \neg shutdown$

e) Context Database Component:

- Safety Property: It is always the case that the system is waiting to receive authenticated information and retrieval requests for database tasks.

$\square\ ((initialize\ )\ \mathcal{W}\ ((DBrequest = 1) \vee \big(\big((DBrequest = 1) \big) \wedge (error = 1) \big))$

- Liveness Property:

$\Diamond\ (initialize = 0 \vee shutdown = 1)\ \wedge$
$\Diamond (initialize = 1) \rightarrow \Diamond (DBrequest = 1)\ \wedge$
$\Diamond\ ((DBrequest = 1) \rightarrow \Diamond (error = 1)) \wedge$
$\Diamond\ (error = 0\ ) \rightarrow \Diamond\ (DBrequest = 1) \wedge$
$\square \Diamond \big(initialize\ \mathcal{W}\ (DBrequest = 1) \big) \vee \neg shutdown$

f) Knowledge-base Component:

- Safety Property: It is always the case that the system is waiting to receive historical context information from context database and retrieve on the stored history.

$\square\ ((initialize\ )\ \mathcal{W}\ ((KBrequest = 1) \vee \big(\big((KBrequest = 1) \big) \wedge (error = 1) \big))$

- Liveness Property:

$\Diamond\ (initialize = 0 \vee shutdown = 1)\ \wedge$
$\Diamond\ (initialize = 1) \rightarrow \Diamond (KBrequest = 1)\ \wedge$
$\Diamond\ ((KBrequest = 1) \rightarrow \Diamond (error = 1)) \wedge$
$\Diamond\ (error = 0\ ) \rightarrow \Diamond\ (KBrequest = 1) \wedge$
$\square \Diamond \big(initialize\ \mathcal{W}\ (KBrequest = 1) \big) \vee \neg shutdown$

g) Payment Management Agent:

- Safety Property: It is always the case that the system is waiting for a management request (e.g. reporting, context-aware payment activation, actual payment request) Figure 5 shows the related state diagram.

$\square\ ((initialize\ )\ \mathcal{W} ((PaymentManagementInfo = 1) \vee (paymentInfo = 1)) \wedge (error = 1))$

- Liveness Property:

$\Diamond\ (initialize = 0 \vee shutdown = 1)\ \wedge$
$\Diamond (initialize = 1) \rightarrow \Diamond \big((PaymentManagementInfo = 1) \vee (paymentInfo = 1) \big)\ \wedge$
$\Diamond\ ((PaymentManagementInfo = 1) \vee (paymentInfo = 1)) \rightarrow \Diamond\ (error = 1)) \wedge$
$\Diamond\ (error = 0\ ) \rightarrow \Diamond \big((PaymentManagementInfo = 1) \vee (paymentInfo = 1) \big) \wedge$
$\square \Diamond\ \big(initialize\ \mathcal{W}\ ((PaymentManagementInfo = 1) \vee (paymentInfo = 1)) \big) \vee \neg shutdown$

h) Payment Knowledge-base:

- Safety Property: It is always the case that the system is waiting to receive historical payment information and retrieve on the stored history.

$$\square\ ((initialize)\ \mathcal{W}\left((PKBrequest = 1) \vee \left(((PKBrequest = 1)) \wedge (error = 1)\right)\right)$$

- Liveness Property: Eventually the component will progress in all stages of its lifecycle.

$$\Diamond\ (initialize = 0 \vee shutdown = 1)\ \wedge$$
$$\Diamond(initialize = 1) \rightarrow \Diamond(PKBrequest = 1)\ \wedge$$
$$\Diamond((PKBrequest = 1)) \rightarrow \Diamond(error = 1))\ \wedge$$
$$\Diamond\ (error = 0\ ) \rightarrow \Diamond\ (PKBrequest = 1)\ \wedge$$
$$\square\Diamond(initialize\ \mathcal{W}\ (PKBrequest = 1)) \vee \neg shutdown$$

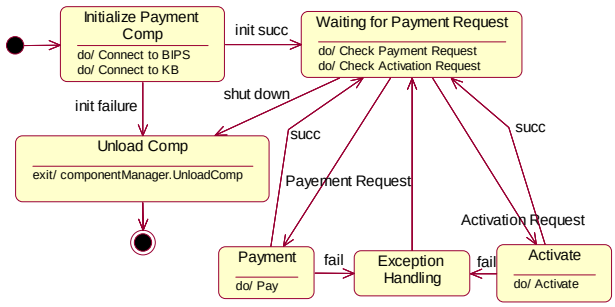

Figure 5: Payment Management Agent Component's State Diagram

Figure 5 illustrates the main states involved in a Payment management Agent component. The components lifecycle is considered in specifying the systems properties. We have expressed both safety and progress requirements of the entire system in terms of mathematical logic. Applying the main properties of CAPSC to verification tools mentioned before, resulted in an acceptable architecture. The result obtained from Hugo/RT was the consistency of our model and its various diagrams. Using the PROMELA output, we verified the properties explained using SPIN as the verifier tool. As a result it is proved that the architecture specified for our payment system is safe to use as the base of developing such systems. Full details of the verification process and its results are available at [12].

V. Conclusions

To overcome potential risks in a supply chain caused by paper-based payments between supply chain members on one hand, and facilities emerged by pervasive and ubiquitous technologies, such as RFID Tags, on the other, we proposed a new payment method using context information that may be called context-aware payment for supply chains. To develop the mentioned method, the payment system's software architecture was specified using Garland & Anthony's approach and UML. This 3-layered architecture insures trusted transactions in a supply chain as well as providing supplier/producer needs automatically. It also used previous known architectures such as SOCAM to provide a context aware payment system. To prove applicability and reliability of the proposed method, certain critical properties of this software architecture such as safety and liveness were verified using Temporal Logic. It was shown that the new method is consistent and safe enough to use. It was also proved that the architecture will eventually progress and overcome potential bottlenecks in the system. This result insists that the proposed architecture, CAPSC, can be the development base for future applications of automatic payment systems according to context information in supply chains.

References

[1] M. Weiser, J. S. Brown, "The Coming Age of Calm Technology," Xeroc PARC, 1996.
[2] M. Baldauf, S. Dustdar, F. Rosenberg, "A Survey on Context-Aware Systems," *International Journal of Ad Hoc Ubiquitous Computing*, Vol. 2, 2007.
[3] S. Gross, R. Muller, M. Lampe, E. Fleisch, "Requirements and Technologies for Ubiquitous Payment," Gallen, Swiss Federal Institute of Technology, 2004, Available: www.inf.ethz.ch/vs/publ/papers/MKWI_UPyament.pdf.
[4] Wikipedia contributors, "Supply Chain," *Wikipedia, The Free Encyclopedia*, March 2008, Available: http://en.wikipedia.org/w/index.php?title=Supply_chain&oldid=198724291.
[5] A. Knox, "Electronic Payment: The Missing Link in Supply Chain Efficiency," *Journal of Financial Transformation-Market Imperfection*, Vol. 14, 2005.
[6] P. Korpipaa, et al., "Managing Context Information in Mobile Devices," *Pervasive Computing, IEEE*, Vol. 2, pp. 42-51, 2003.
[7] T. Hofer, et al., "Context-Awareness on Mobile Devices - The Hydrogen Approach," in *Proceedings of 36th Annual Hawaii International Conference on System Science*, pp. 292-302, 2003.
[8] H. Chen, "An Intelligent Broker Architecture for Pervasive Context-Aware Systems," University of Maryland, Baltimore County, 2004, PhD Thesis.
[9] D. Selber, A.K. Dey, and G.D. Abowd, "The Context Toolkit: Aiding the Development of Context-Aware Applications," In Proceedings of Human Factors in Computing Systems: CHI 99, ACM Press, pp. 434-441, 1999.
[10] T. Gu, H.K. Pung, and D.Q. Zhang, "A Service-Oriented Middleware for Building Context-Aware Services," *Journal of Network and Computer Applications*, Vol. 28, pp. 1-18, 2005.
[11] J. Garland, and R. Anthony, *Large-Scale Software Architecture: a Practical Guide Using UML*, John Wiley and Sons, 2003.
[12] Z. Zamani, M. Bayat, A. Moeini, "Technical Report on the CAPSC Architecture Using UML and Verification Tools," University of Tehran, 2008.
[13] C. M. Holloway, "Why Engineers Should Consider Formal Methods," *16th Digital Avionics Systems Conference,* 1997.
[14] M. Fisher, M. Wooldridge, "On the Formal Specification and Verification of Multi-Agent Systems," *IJCIS*, vol. 6, pp. 37.65, 1997.
[15] Z. Manna, A. Pnueli, "The Temporal Logic of Reactive and Concurrent Systems," *Springer-Verlag*, 1992.
[16] S. Gnesi, F. Mazzanti, "A Model Checking Verification Environment for UML Statecharts," *in Proceedings of XLIII Congresso Annuale*, Udine, Italy, October 2005.
[17] J. Jeffrey, P. Tsai, K. Xu, "A comparative study of formal verification techniques for software architecture specifications," *Annals of Software Engineering, Springer,* p.p. 207-223, 2000.
[18] M.E. Beato, M. Barrio-Solorzano, C.E. Cuesta, "UML Automatic verification tool (TABU)", in *Proceedings of Specification and Verification of Component-Based Systems (SAVCBS 04)*, California, USA, 2004.
[19] D. Latella, I. Majzik, and M. Massink, "Automatic Verification of a Behavioural Subset of UML Statechart Diagrams Using the SPIN Model-Checker," *Formal Aspects of Computing, Springer*, 1999.
[20] S. Gnesi, "Formal Specification and Verification of complex systems," *Electronic Notes in Theoretical Computer Science, Elsevier*, 2003.
[21] Z. Manna, "STeP: The Stanford Temporal Prover (Educational Release), User's Manual," Technical report, Computer Science Department, Stanford University, November 1995.
[22] "SMV Tutorial," McMillan Cadence Berkeley Labs, Available:
[23] http://www.cs.indiana.edu/classes/p515/readings/smv/CadenceSMV-docs/smv/tutorial/tutorial.html
[23] UPPAAL Home, Available: http://www.uppaal.com.
[24] Official Site on SPIN: the on-the-fly Model checker, Available: http://www.spinroot.com.
[25] A. Knapp, and S. Merz, "Model checking and code generation for UML state machines and collaborations," in *Proceedings of 5th Workshop on Tools for System Design and Verification (FM-TOOLS'02)*, 2002.
[26] A.P. Sistla. "Safety, Liveness and Fairness in Temporal Logic," *Formal Aspects of Computing, Springer*, 1994.