MODELING INTERACTIVE SYSTEMS WITH HIERARCHICAL COLORED PETRI NETS

Mohammed Elkoutbi and Rudolf K. Keller
Université de Montréal, DIRO, C.P. 6128, Succursale Centre-ville, Montréal, Canada, H3C 3J7
{elkoutbi, keller}@iro.umontreal.ca <http://www.iro.umontreal.ca/~{elkoutbi, keller}>

KEYWORDS

Formal models, hierarchical colored Petri nets, UML, scenario integration

ABSTRACT

This paper addresses the problem of modeling interactive systems*. We aim to provide an easy way for elaborating the behavior specification of a system using hierarchical colored Petri nets. Our modeling approach comprises two levels of abstraction: the use case level corresponding to the use case model as defined in the Unified Modeling Language (UML) and the scenario level as refinement of the former one. The color aspect of nets is used at the scenario level to preserve the independence of several scenarios after their integration.

The benefits of our approach consist in the structuring of the scenario acquisition and in the new approach of merging them using colors and chameleon tokens.

1. INTRODUCTION

The need for formal techniques for analyzing systems is widely acknowledged, a large range of existing formalisms being in use for specifying systems. In modeling interactive systems visual formalisms are needed to reduce the gap between users and analysts. Object-oriented methods like UML (Booch et al. 1997) offer one of these formalisms (Statecharts), yet they only address the dynamic behavior of individual objects. The behavior of the overall system can not be described explicitly; it must be synthesized from the Statecharts of the objects of the system.

The main contribution of this paper is to provide an approach for the formal specification of the behavior of an overall system. We propose a process for deriving system specifications combining the UML as object-oriented method and colored Petri nets as formal method. At the beginning of the process, we elaborate a use case diagram according to the UML. Then we transform this diagram into a first Petri net having use cases as places and user interactions as guard conditions of transitions. Each place of this first net (use case) is then refined by a colored Petri net constructed from the scenarios associated with the various use cases. The choice of colored Petri nets as formalism was guided by natural support for concurrency and the notion of colored tokens that is crucial in scenario integration. The use of colored Petri net tools may allow for verification and simulation of the resulting specifications. Especially, several use cases may be executed concurrently, and several scenarios and several copies of the same scenario may be simulated. Two further contributions of our work is a new algorithm for integrating scenarios and the notion of chameleon token, introduced in order to have specifications capture exactly the scenarios of the system and nothing more.

In this paper, we first introduce in section 2 the formalism used to describe Petri nets, as a basis for describing our integration algorithm. In section 3, we provide an overview of the UML, with a special focus on use case diagrams and sequence diagrams. Then, in section 4, we present the process for deriving system specification. Section 5 presents related work, and section 6 concludes this paper.

2. PETRI NETS AND COLORED PETRI NETS

Petri Nets (PN) are in use in a large variety of different areas. Their application ranges from informal to formal systems and from software to hardware systems and from sequential to concurrent systems. As mentioned in (Jensen 1997) Petri nets are used in communication protocols, distributed algorithms, computer architecture, computer organization, human-machine interaction and many others areas.

A simple PN is defined as a bipartite graph consisting of places, transitions and tokens. PN = <P, T, A, M>. P: the set of places, T: the set of transitions, A: the set of directed arcs connecting places and transitions and M: the set of tokens resident in places at a given moment. In addition, a

* This research is supported by FCAR, (Fonds pour la Formation de Chercheurs et l'Aide à la Recherche au Quebec) and NSERC (Natural Sciences and Engineering Research Council of Canada).

net may have an associated set of enabling and firing rules to determine under what conditions (particular marking) a transition is enabled and may fire.

In real-world systems, we often find many parts that are similar. These parts must be represented by disjoint and identical sub-nets in PNs. This means that the net becomes largely and it becomes difficult to see the similarities between the individual sub-nets. CPNs provide a more compact representation where individual sub-nets are replaced by one sub-net with different kind of tokens, each token having a color and representing a different sub-net in the equivalent PN.

There are many various definitions of CPNs. In our work, we use Lakos's definition (Lakos 1994), which is slightly different from Jensen's formulation (Jensen 1997).

Some definitions to understand for the CPN formalization:

a) the set of elements of type T can be denoted T.
b) the multi-set of elements of type T is denoted T*.
c) multi-set addition, subtraction, scalar multiplication, comparison operations are denoted in the usual way (m1 – m2, m1 <= m2, etc.)
d) the type of a variable v is denoted Type(v).
e) the type of an expression expr is denoted Type(expr).
f) The set of variables in expression expr is denoted Var(expr).
g) The binding b of a set of variables V, where $\forall v \in V$: $b(v) \in Type(v)$

A colored Petri net (CPN) is defined as a tuple $<\sum, D, P, T, A, \tau, G, E, I>$ where:

$\sum$ is a finite set of non-empty types, called color sets.

D is a finite set of data fields.

P is a finite set of places with $P \subseteq D$.

T is a finite set of transitions with $D \cap T = \emptyset$.

A is a finite set of arcs such that $A \subseteq P \times T \cup T \times P$.

τ is a color function, $\tau: D \rightarrow \sum$, $\forall p \in P$, $\tau(p) = C^*$ and $C^* \in \sum$.

G is a guard function, $G: T \rightarrow expr$ where:

$\forall t \in T$: $[Type(G(t)) = bool \wedge Type(Var(G(t))) \subseteq \sum]$

E is an arc expression function, $E: D \times T \cup T \times D \rightarrow expr$ where:

$E(x1, x2) = \emptyset$ if $(x1, x2) \notin A$ and

$\forall a \in A$: $[Type(E(a)) = \tau(p(a)) \wedge Type(Var(E(a))) \subseteq \sum]$

p(a) is the place of arc a.

I is an initialization function, $I: D \rightarrow expr$, where

I(d) is a closed expression and $\forall d \in D$: $[Type(I(p)) = \tau(p)]$.

In this definition, arcs expressions specify tokens which are being added or removed by transitions. Both places and variables of expressions are typed, and the initial marking is defined by the initialization function I.

3. THE UNIFIED MODELING LANGUAGE

The UML represents the unification of the best-known object-oriented methodologies, to provide a standard in the domain of object-oriented analysis and design. The UML does not provide a process for developing software, but it gives a syntactic notation, to describe all parts of a system (data, function and behavior) through a number of diagrams (class diagram, use case diagram, sequence message diagram, collaboration diagram, state diagram, activity diagram, implantation diagram and deployment diagram).

The use case diagram in UML presents a collection of use cases and the external actors whose interact with the system. A use case is a generic description of an entire transaction involving several objects of the system. Use cases are represented as ellipses, and actors are depicted as icons connected with solid lines to the use cases which they interact with. One use case can call upon the services of another use case. Such a relation is called a *uses* relation and is represented by a directed dashed line. The direction of a *uses* relation does not imply order of execution. Figure 1 shows an example of a use case diagram corresponding to an automatic teller machine (ATM). There is one actor (customer) interacting with four use cases (*Identify*, *Deposit*, *Withdraw*, and *Balance*), the use case *Print* for example is used by three use cases (*Identify*, *Deposit*, and *Withdraw*).

An execution (instance) of a use case is called a scenario, there might be many possible scenarios for one use case. Scenarios can be represented by sequence diagrams or collaboration diagrams, and conversion between these diagrams is possible. We have chosen to represent scenarios by sequence diagrams for their simplicity and their wide use. In the rest of this paper we will call them scenario diagrams.

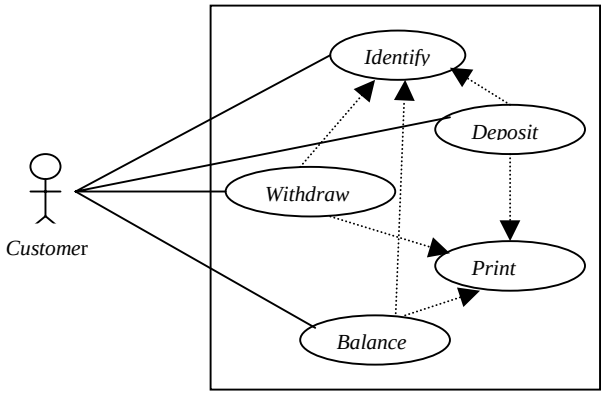

Figure 1: Use case diagram of ATM

A scenario diagram shows interaction among a set of objects in temporal order. In Figure 2, we give an example of two scenarios corresponding to the use case *Identify* of the ATM system.

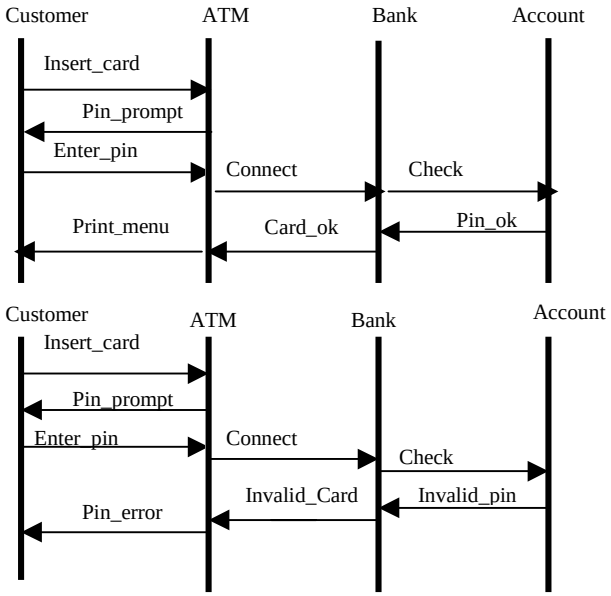

Figure 2: Example of two scenario diagrams corresponding to the use case *Identify*.

4. DESCRIPTION OF APPROACH

In this section, we describe the process for obtaining the formal specification of the behavior of the system to be modeled. Note that we address the behavior of the entire system and not just the behavior of its constituent objects. For this purpose, we have used in our approach the two kinds of UML diagrams described in the previous section and combined them with colored Petri nets. The approach is a four-step process:

1. the elaboration of the use case diagram of the system, and the generation of the corresponding PN ;
2. the refinement of uses relations and the updating of the PN of step 1;
3. the elaboration of several scenarios for each use case;
4. the integration of the scenarios by use case;

4.1. Elaboration Of The Use Case Diagram

We begin by elaborating the use case diagram of the system, and then we derive the first PN where each use case is transformed into a place of the PN. The transition leading to this place is guarded by a condition corresponding to the initiating of the use case by the actor. We add in the generated PN a place Begin modeling the entry of the system (initial point), and after execution of the use cases, the system returns to its initial place Begin. This place will contain several tokens for modeling the concurrent execution of use cases. For example, Figure 3 shows the derived PN for an ATM system. We use a unique prefix to distinguish the guard conditions of the different use cases. In the example below we are using the first letter to refer to a specific use case ('W initiated' means the use case *Withdraw* initiated, and 'W exited' means the end of the use case *Withdraw*).

4.2. Refinement Of Uses Relation

In the use case diagram, a use case can call upon the services of another with the relation uses. This relation may have several meanings depending on the system. Consider two use cases Uc_1 and Uc_2, the relation uses between them may be interpreted in different ways. Figure 4(a) gives the general form of this relation (Note that in use case diagrams, control constructs such as loops and conditions which would lead to a more general form of this relation, are not considered). Uc_1 is decomposed into three sub-use cases: Uc_{11} represents the part of Uc_1 executed before the call of Uc_2, Uc_{12} is the part executed concurrently with Uc_2, and Uc_{13} is the part executed after termination of Uc_2 (synchronization). It is possible that two of these three sub-use cases are empty, resulting in one of the configuration types shown in Figure 4(b), Figure 4(c), Figure 4(d), Figure 4(e), Figure 4(f) and Figure 4(g).

A relation of type (g) between Uc_1 and Uc_2 means that Uc_2 precedes Uc_1, this implies that Uc_1 is not directly accessible from the place *Begin*. So transitions form *Begin* to Uc_1 must be changed to transitions from Uc_2 to Uc_1.

In the ATM system (Figure 1), the relation uses between (*Deposit*, *Identify*), (*Withdraw*, *Identify*) and (*Balance*, *Identify*) are of type (g), and the relations between (*Deposit*, *Print*), (*Withdraw*, *Print*) and (*Balance*, *Print*) are of type (e). Figure 5 shows the updated PN representing the refinement of the *uses* relations.

4.3. Scenario Elaboration

As a result of step 2, we obtain a list of elementary use cases denoted $\{Uc_1, Uc_2, \ldots, Uc_n\}$. For each use case Uc_i the analyst acquires the related scenarios $\{Sc_{i1}, Sc_{i2}, \ldots, Sc_{im}\}$. The acquisition of a scenario Sc_{ik} comprises the construction of the scenario diagram and the associated table of object states. This table is derived from the scenario diagram by following the exchange of messages from the top to the bottom and identifying the changes in object states after sending the messages. Table 1 gives as an example the object states table, associated to the first scenario of the use case *Identify* described in Figure 2. In this table a scenario state is represented by the union of the states of the objects participating in the scenario.

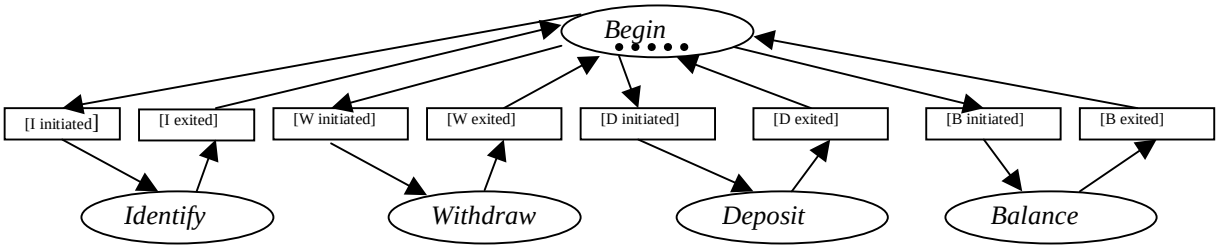

Figure 3: PN corresponding to the ATM system.

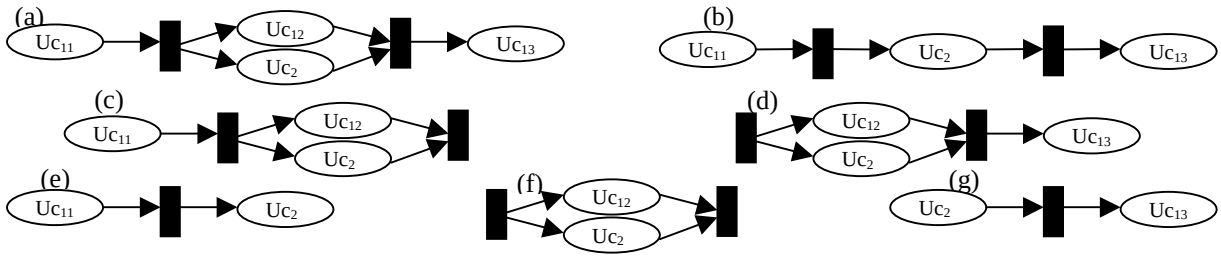

Figure 4: Refinement of the relation uses.

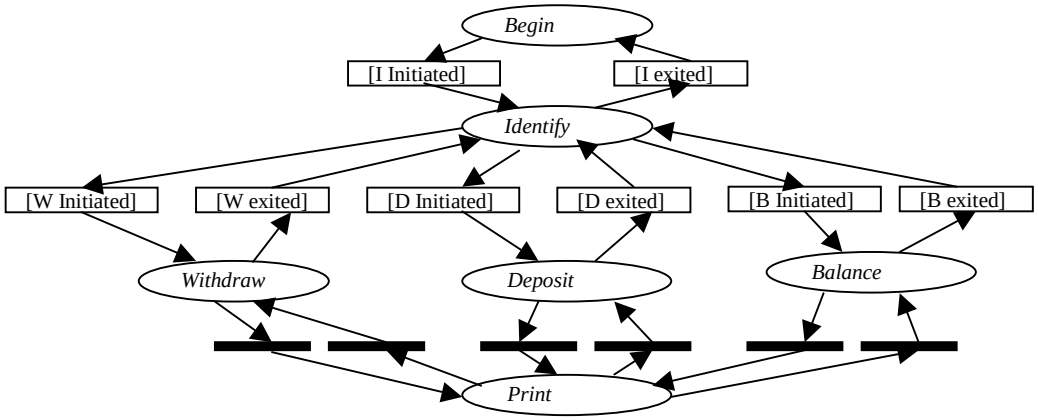

Figure 5: Updated PN after refinement of the relation uses.

From this table, we obtain a CPN associated to the scenario by transforming scenario states into places and messages into guard conditions for transitions. Figure 6 shows the CPN for the first scenario of the *Identify* use case. We give the same color to all places in the obtained CPN, whith place B representing the beginning of the use case. All scenarios related to the same use case have the same initial place but different colors.

4.4. Scenario Integration

In this step, we aim to merge several CPNs corresponding to all scenarios of a use case $Uc_{i,}$ to produce a new CPN modeling the behavior of the use case as a whole. The proposed algorithm provides an incremental way for integrating scenarios. If we consider two scenarios Sc_1 and Sc_2 for a given use case and their corresponding CPNs, the algorithm will merge places in Sc_1 and Sc_2 having the same scenario state into the same place. The merged place will have as color the union of colors of the two scenarios. For transitions, the algorithm looks for transitions in the two scenarios that have the same input and output places and merges them with an OR between their condition guards. In the following, we give a formal description of the algorithm. We use // for delimiting comments.

$Sc_1 = <\sum 1, D_1, P_1, T_1, A1, \tau 1, G1, E1>$ where:

$\sum 1 = \{c1\}$,

$\tau 1$: $D1 \rightarrow \{c1\}$,

G1 is formed by conditions associated to messages,

E1 is the identity function.

$Sc_2 = <\sum 2, D_2, P_2, T_2, A2, \tau 2, G2, E2>$

Objects Messages	Customer	ATM	Bank	Account	Scenario state
Insert_card	Present	Card_in			S1 = {Present, Card_in}
Prompt_pin	Present	Wait_pin			S2 = {Present, Wait_pin}
Enter_pin	Present	Pin-entered			S3 = {Present, Pin_entered}
Connect	Present	Pin-entered	Connected		S4 = {Present, Pin_entered, Connected}
Check	Present	Pin-entered	Connected	Checked	S5 = {Present, Pin_entered, Connected, Checked}
Pin_ok	Present	Pin-entered	Valid_pin	Checked	S6 = {Present, Pin_entered, Valid_pin, Checked}
Card_ok	Present	Valid-card	Valid_pin	Checked	S7 = {Present, Vaild_card, Valid_pin, Checked}
Print_menu	Present	Menu	Valid_pin	Checked	S8 = {Present, Menu , Valid_pin, Checked}

Table 1: Object states associated to the first scenario of *Identify*.

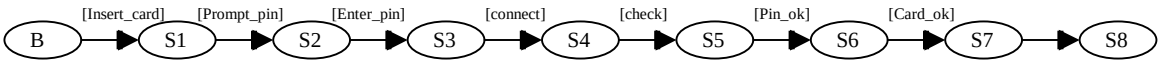

Figure 6: CPN corresponding to Identify scenario.

$\sum 2 = \{c2\}$,
$\tau 2: D2 \rightarrow \{c2\}$.

$Sc = <\sum, D, P, T, A, \tau, G, E>$
$\sum = \{\{c1\}, \{c2\}, \{c1,c2\}\}$,
$D = D1 \cup D2$,
$P = P1 \cup P2$,
$\tau: D \rightarrow \sum$ where
$\forall p \in P1 \cap P2 : \tau(p) = \{c1,c2\}$,
$\forall p \in P1 \setminus (P1 \cap P2) : \tau(p) = \{c1\}$ and
$\forall p \in P2 \setminus (P1 \cap P2) : \tau(p) = \{c2\}$.

The construction of the resulting set of transitions T, is given by the following algorithm:

Begin
T = ∅
For each t ∈ T1
calculate •t and t•
// •t = {p|(p, t) ∈ A} and t• = {p|(t, p) ∈ A}
If ∃ t' ∈ T2 | •t' = •t and t'• = t• then // merge
T = T ∪ {t}
T2 = T2 \ {t'}
G(t) = G1(t) ∨ G2(t')
delete t' from A2, G2 et E2
Else
T = T ∪ {t}
G(t) = G1(t)
Endfor
For each t ∈ T2 // remainder transitions in T2
T = T ∪ {t}
G(t) = G2(t)
Endfor
// calculation of A
A = A1 ∪ A2
// calculation of E
For each a ∈ A having the form (p,t)
Type(E(a)) = Type(τ(p))
For each a ∈ A having the form (t,p)
Type(E(a)) = Type(τ(p)) ∩ Type(τ(•t))
End.

In general, after integrating several scenarios, the resulting specification captures the initial scenarios and perhaps even more. Figure 7 illustrates this problem. The resulting scenario Sc will capture Sc1 (B,S1,S2,S3,S4,S5), Sc2 (B,S1,S6,S3,S7,S5) and two other scenarios (B,S1,S2,S3,S7,S5) and (B,S1,S6,S3,S4,S5). In the initial place B of Figure 7, what color will have the token? If it has c_1 (respectively c_2) as color that means that we have chosen to execute Sc_1 (respectively Sc_2). But in this place we do not know which scenario the user will execute. For solving this problem, we have introduced a "chameleon token" (token with several colors). As soon as it visits the places of the integrated net, depending on the sequence of transitions fired, it will be marked by the intersection of its colors and the colors of the place visited. In Figure 7, we represent the chameleon token with a white color (similar to the white color of light which is in reality composed of several monochromatic colors). When this token passes to the place S_1, it stays multi-color $\{c_1, c_2\}$, and if it passes from S_1 to S_2, its color changes to c_1 and keeps it along the rest of the net, or if it passes from S_1 to S_6, its color changes to c_2 and keeps it along the rest of the net.

5. RELATED WORK

In the area of scenario integration, most research has only addressed the problem of sequential integration (Koskimies and Makinen 1994; Some et al. 1995; Desharnais et al.

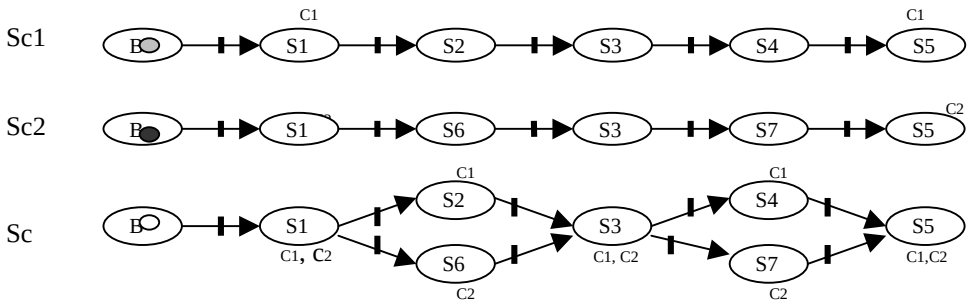

Figure 7: Problem of interleaving between scenarios.

1997), and few researchers have been interested in a more general form of integration.

Koskimies (Koskimies and Makinen 1994) presents an algorithm for synthesizing a Statechart for an object of a system from a list of scenarios. He infers a Statechart that is able to execute all traces corresponding to the input scenarios. Desharnais (Desharnais et al. 1997) defines a scenario as the union of two relations Re and Rs where Re represents the relation of the environment which captures all the possible actions of the environment and Rs the relation corresponding to the system reaction. The scenario integration is given by the composition of the scenarios relations. Glinz (Glinz 1995) gives a way for composing scenarios represented by Statecharts using some operators (conditional, iterative and concurrent) , but without supporting scenarios overlapping. Dano (Dano et al. 1997) has proposed a formalization of use cases with Petri nets, he defines a list of temporal relations between use cases (begin at the same time, end at the same time, one after the other, etc.).

6. CONCLUSION

In this paper, we have proposed an new approach for elaborating a specification of the system behavior by using two kinds of Petri nets: a simple Petri net modeling the relation between use cases linked to several colored Petri nets representing the use cases behavior. We have also given an new algorithm for scenario integration which merges several scenarios corresponding to the same use case and preserves the independency between these scenarios after integration by means of colors.

There are several advantages with the proposed approach. First, the process of scenario acquisition is more structured. In the work described in the previous section, the user gives the analyst the scenarios without any guidance, covering the entire system or covering several tasks (use cases), or representing a part of a use case. In our work, we acquire scenarios use case by use case which we consider as more natural. Second, our approach solves the problem of interleaving scenarios by means of colors and chameleon tokens. Finally, as a consequence of using Petri nets, our approach allows modeling of concurrency between use cases, between scenarios and between copies of the same scenario.

As future work, we plan to extend our work by deriving a specification for interface objects and generating a prototype of the user interface. The verification aspect will also be addressed, in order to provide an incremental method for verifying the entire system, for example to consider verifications at the two levels of detail (use case level and scenario level). Finally, the impact of chameleon token on existing methods for verification and simulation will be studied.

BIBLIOGRAPHY

Dano, B.; Briand, H. and Barbier, F. 1997. “An Approach Based on the Concept of Use Cases to Produce Dynamic Object-Oriented Specifications.” *In Proceedings of the Third IEEE International Symposium on Requirements Engineering* (RE’97).

Desharnais, J.; Khediri, R.; Frappier, M. And Mili, A. 1997. “Integration of Sequential Scenarios.” *In Proceedings of the Sixth European Software Engineering Conference* (ESEC’97), 310-326.

Glinz, M. 1995. “An Integrated Formel Model of Scenarios Based on Statecharts” *In fifth European Software Engineering Conference*, Lecture Notes in Computer Science vol. 989, Springer-verlag, 254-271.

Jensen, K.1997. *Coloured Petri Nets, Basic concepts, Analysis methods and Pratical Use*. Springer .

Koskimies, K. And Makinen, E. 1994. “Automatic Synthesis of State Machine from Trace Diagrams.” *Software Practice & Experience*, vol. 24, no. 7, 643-658.

Lakos, C. 1994. “Definition and Relationship to Coloured Nets.” Technical Report TR94-3, Department of Computer Science, University of Tasmania.

Some, S.; Dssouli, R. and Vaucher, J. 1995. “From Scenarios to Timed Automata: Building Specifications from Users Requirements.” *In Proceeding of the 2nd Asia Pacific Software Engineering Conference*.

Booch, G.; Rumbaugh, J. And Jacobson, I. 1997. “The Unified Modeling Language for object-oriented development”, documentation set version 1.0, Rationale Software Corporation, Santa Clara, CA.