Real-time system specifications based on UML Scenarios and Timed Petri Nets

Mohammed Elkoutbi[1], Mohamed Bennani[1], Rudolf K. Keller[2] and Mohamed Boulmalef[3]

1 *École Nationale Supérieure d'Informatique et d'Analyse des Systèmes, Agdal, Rabat, Morocco. elkoutbi@ensias.ma*

2 *Zühlke Engineering AG, Zürich-Schlieren, Switzerlan. ruk@zuehlke.com*

3 *United Arab Emirates University. boulmalf@uaeu.ac.ae*

Abstract. In this paper, we suggest a requirement engineering process for real-time systems that yields a formal specification of the system using timed Petri nets. Scenarios are acquired in form of sequence diagrams as defined by the Unified Modeling Language (UML), and are enriched with time constraints information. These diagrams are transformed into partial timed Petri net specifications and then merged to obtain a global timed Petri net specification capturing the behavior of the entire system.

1 Introduction

Scenarios are identified as a promising technique for understanding requirements [And93] and system specification [Hsi94]. A typical process for requirement engineering based on scenarios such as proposed by Hsia et al. [Hsi94], presents two main tasks. The first task consists to generate specifications that describe system behavior form scenarios. The second task concerns scenario validation with users by using a system prototype. However, these tasks remain tedious activities, since, we lack for algorithmic support and tools that automate them.

Object-oriented methods offer a good framework for scenarios [Boo94, Jac92, Rum91]. We have used in this work the UML that emerges as a unified notation for object-oriented analysis and design. It gives notations for describing a system in various views, but it does not define any specific process for software development, beyond some preliminary process description reported, for instance, in [Jac99].

This paper suggests a requirement engineering approach for real-time systems. It provides a three activities process for synthesizing a timed Petri net specification of the system from UML scenarios. At the first step of the process, we elaborate the use case diagram according to the UML. Each use case is then refined by the elicitation of the scenarios associated. Scenarios are acquired using UML sequence diagrams enriched with time constraints information. These diagrams are transformed into partial timed Petri net specifications modeling scenarios behavior, and then merged to obtain a compact specification modeling the entire system behavior.

Section 2 of this paper presents the timed Petri net specifications used in this work. Section 3 gives a brief overview of the UML, with a special focus on sequence diagrams. Then we present in Section 4, the process for deriving specifications from scenarios. Section 5 presents some related work, Section 6 concludes this paper.

2 Timed Petri Nets

Several Petri Net (PN) models were proposed to support time. Simple value of time was associated to transitions, to places and arcs [Ram73]. The behavior of communication systems that depend generally of time-out, needs a more complex model of time. Petri nets models that associate an interval of time to transitions will be more suitable for this kind of systems. In this work, we use the following formulation of Timed Petri Nets (TPN) [Ram80].

TPN = <PN, Temp> where:
PN is a simple Petri net = <P, T, A, M0>;
P: set of places; T: set of transitions;
A: set of arcs; M0: initial marking;
Temp: $T \rightarrow Q^{+} \cup \{0\} \times Q^{+} \cup \{\infty\}$;
Q^{+} is the set of rational positive numbers;
Temp(t)=[a, b] (0<=a<=b);

The transition t will be valid at the time a, and it will be fired at most at time b. b-a represents the maximal duration of the transition t.

3 Unified Modeling Language

The UML [Rum99] is a rich and powerful language that can be used for problem conceptualization, software system specification, as well as implementation. In this section, we discuss the UML diagrams that are relevant of our approach: Sequence diagrams (SequenceD). Even though the UML provides Statechart diagrams to specify objects behaviors, we have used TPNs because of their support of pure concurrency and time, their ability of specifying and simulating multiple copies of scenarios in the same specification, and the wide range of

existing tools for analyzing, simulating, and verifying Petri net specifications.

3.1 Sequence Diagram (SequenceD)

A SequenceD shows interactions among a set of objects in temporal order, which is good for understanding timing issues. In particular, it shows the objects participating in the interaction by their lifelines and the messages that they exchange arranged in time sequence. Usually only time sequences are important but in real-time systems the time axis could be a metric. A message may be guarded by a condition, annotated by an iteration, and/or constrained by an expression. Constraints are used in this work to enrich messages with time constraints information. The figure Fig.1 shows all possible constructs that we can have in UML. The number preceding messages indicates the sequence order. The star (*) indicates an iterative behavior and letters like A and B in the example of Fig.1 show concurrent behaviors.

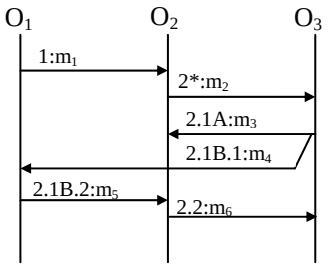

Fig.1: Interaction constructs in SequenceDs.

4 Description of the approach

In this section, we describe the proposed iterative process for requirements engineering.

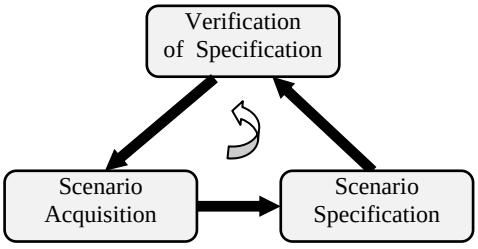

Fig.2: Requirements engineering based on scenarios

In the *Scenario Acquisition* activity (Fig.2), the analyst elaborates the UsecaseD, and for each use case, he or she elaborates its corresponding scenarios in form of SequenceDs. The *Scenario Specification* activity consists of deriving Timed Petri Nets (TPN) from the acquired UsecaseD and SequenceDs. Partial TPNs corresponding to are iteratively merged to obtain an integrated TPN that serves as input to the *Verification of specification* activity where existing tools and techniques are used.

4.1 Scenario acquisition

In this activity, the analyst acquires scenarios as enriched SequenceDs for each use case in the UsecaseD. The extension of SequenceDs is made to support time constraints in UML.

Beyond the UML standard message constraints found in SequenceDs, we define eight additional constraints to support time constraints. Note that the UML defines two standard constraints for messages: *vote* and *broadcast*. The *vote* constraint restricts a collection of return messages, and the *broadcast* constraint specifies that the constrained messages are not invoked in any particular order.

UML constraints are generally put between braces. After studying some related work on scenarios that formalize time constraints [Dan97, Sal02], we propose to extend the UML with the following message constraints:

Constraint	significance
m{At(a)}	the message m will occur at the time a
m{After(a)}	the message m will occur after the time a
m{Before(b)}	the message m will occur before the time b
m{Between(a,b)}	the message m will occur at the time a, and will takes at most b-a seconds
m_1{Starts(m_2)}	the messages m_1 and m_2 start at the same time
m_1{Ends(m_2)}	the messages m_1 and m_2 finish at the same time
m_1{Equals(m_2)}	the messages m_1 and m_2 start and finish at the same time
m_1{Meets(m_2)}	m_1 starts before the end of m_2

The four first constraints are applied to single messages, while the four others indicate time constraints between two messages. This range of time constraints gives a good modeling framework for communication systems.

The figures Fig.3 and Fig.4 give examples of SequenceDs annotated with time constraints corresponding to two scenarios of the telephone system. The first scenario represents the case where the user B is idle (Fig.3), the second shows interactions where the user B is busy (Fig.4).

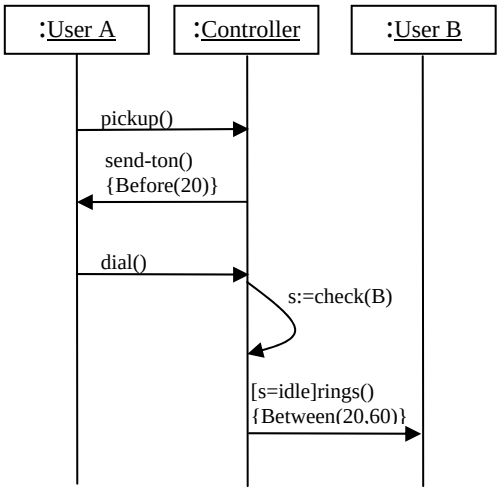

Fig.3 : Scenario where the user B is idle.

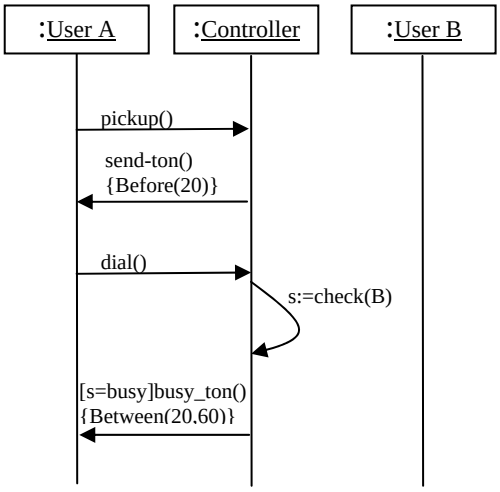

Fig4. : Scenario where the user B is busy.

4.2 Scenario specification

This activity consists of two main tasks: deriving partial TPN specifications from enriched UML SequenceDs and integrating partial derived specifications into a global specification capturing the behavior of the entire system.

4.2.1 Deriving TPNs from SequenceDs

For each scenario of a given use case, we derive the structure and semantic of the equivalent TPN. The TPN structure is obtained from the graph representing the sequence of messages in the scenario by adding places between each pair of sequential messages. Figure Fig.5(a) gives an example of such graph derived from the scenario of the Figure Fig.1, and Figure Fig.5(b) shows the inserted places.

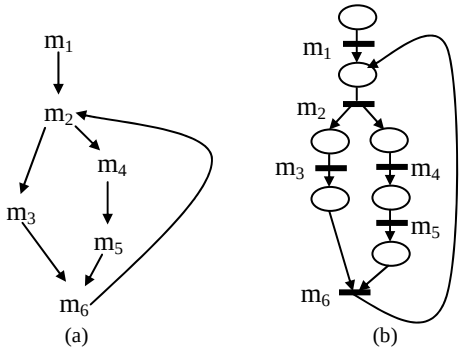

Fig.5: (a) Sequence of messages derived from the scenario of Fig.1.
(b) Updated sequence by inserting places.

For the TPN semantic, the analyst adds labels to places. This operation can be automated by putting transitions pre-conditions as place labels.

Time constraints can be easily transformed into time intervals associated to transitions in the equivalent TPN using the following rules:

At(a) → [a,∞].
After(a) → [a,∞]
Before(b) → [0,b].
Between(a, b) → [a, b].

$m_1\{Starts(m_2)\}$ → $[a_1, b_1]$ of m_1, $[a_2, b_2]$ of m_2 and $a_1=a_2$
$m_1\{Ends(m_2)\}$ → $[a_1, b_1]$ of m_1, $[a_2, b_2]$ of m_2 and $b_1=b_2$
$m_1\{Equals(m_2)\}$ → $m_1\{Starts(m_2)\}$ and $m_1\{Ends(m_2)\}$
$m_1\{Meets(m_2)\}$ → $[a_1, b_1]$ of m_1, $[a_2, b_2]$ of m_2 and $a_1<b_2<b_1$

Applying the above transformations to SequenceDs of Figures Fig.3 and Fig.4, we obtain respectively the TPNs shown in figures Fig.6(a) and Fig.6(b).

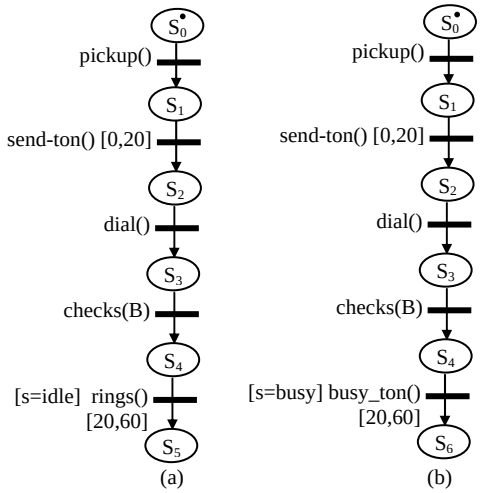

Fig.6: Obtained TPNs from scenarios of figures Fig.3 and Fig.4.

4.2.2 Integrating TPNs specifications

The algorithm behind this operation, is based on a preliminary version presented in [Elk98]. It takes an incremental approach to integration. Given two scenarios with corresponding TPNs TPN_1 and TPN_2, the algorithm merges all places in TPN_1 and TPN_2 having the same names. Then, the algorithm looks for transitions having the same input and output places in the two scenarios and merges them with an *OR* between their guard conditions.

Suppose that:

$TPN_1 = <P_1, T_1, A_1, M0_1, Temp_1>$

$TPN_2 = <P_2, T_2, A_2, M0_2, Temp_2>$

$TPN = <P, T, A, M0, Temp>$ is the resulting TPN

$P = P_1 \cup P_2$

$A = A_1 \cup A_2$

$T = T_{i1} \cup T_c \cup T_{i2}$ where:

$Tc = \{t \in T_1 \cap T_2 / TPN_1(\bullet t) = TPN_2(\bullet t)$ and $TPN_1(t\bullet) = TPN_2(t\bullet)\}$

$Ti_1 = T_1 \setminus Tc$ (\ is the difference between sets)

$Ti_2 = T_2 \setminus Tc$

$\forall p \in P, M0(p) = M0_1(p) + M0_2(p)$

$\forall t \in T_1, Temp(t) = Temp_1(t)$

$\forall t \in T_2, Temp(t) = Temp_2(t)$

$\forall t \in T_c, Temp(t) = Temp_1(t) \cup Temp_2(t)$ when $Temp_1(t) \cap Temp_2(t) \neq \varnothing$

When $Temp_1(t) \cap Temp_2(t) = \varnothing$, an error is displayed indicating existing incoherence between the two scenarios.

Applying this integration algorithm to TPNs of figure Fig.6, we obtain the integrated TPN shown in figure Fig.7.

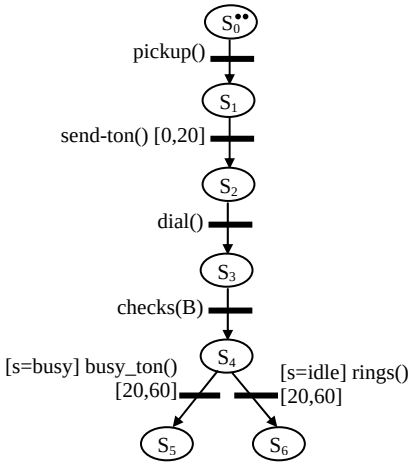

Fig.7: Integrated TPN of TPNs given in Fig.6.

4.3 Verifying integrated TPN specification

For timed Petri nets, two approaches are used for verification issues: enumerative and structural analysis. The first approach [Rou85, Ber91] is interested to the covering tree, while the second [Rou85] is more interested by dynamic and performance aspects.

In addition to these abundant algorithms and tools, incoherence relating to temporal constraints can be detected while integrating scenarios.

A transition with timed constraints that belongs to two scenarios, must verify the following rules prior integration:

Scenario1	Scenario2	Rule
After(a1)	At(a2)	a2>a1
Before(a1)	At(a2)	a2<a1
After(a1)	Before(a2)	a2<a1
Between(a1,b1)	At(a2)	a1<a2<b1
Between(a1,b1)	After(a2)	a2<b1
Between(a1,b1)	Before(a2)	a2<b1
Between(a1,b1)	Between(a2,b2)	a2<b1 and b2>a1 or a1<b2 and b1>a2

Semantically, these rules are equivalent to the conditions discussed in the previous section (When $Temp_1(t) \cap Temp_2(t) = \varnothing$, an error is displayed indicating existing incoherence between the two scenarios).

5 Related Work

In the area of scenario integration, most research has only addressed the problem of sequential integration, and few researchers have been interested to real-time systems scenario integration [Sal02, Del97].

Salah et al. [Sal02] represent a scenario as a timed automaton. He defines a scenario integration algorithm to generate a timed automaton modeling the behavior of the entire system.

Delatour et al. [Del99] proposes a way to combine UML with Petri nets to model real-time systems. He also enhances SequenceDs to support time constraints. No integration algorithm is provided in his approach.

Our approach meets more the Delatour's one with in more algorithmic support for scenario integration. It provides an incremental way to integrate real-time scenarios to produce a formal description of the system in form of timed Petri nets. This formalism supports natural concurrency and gives us the possibility to

simulate more than one scenario at a time even multiple copies of the same scenario at a time.

6 Conclusion

In this paper, we have proposed an iterative process for elaborating a system specification using UML scenarios and timed Petri nets. We propose to extend UML scenarios with real time constraints. We provide an algorithmic support to transform SequenceDs into timed Petri nets and to then integrate the obtained partial scenario specifications into a more compact one.
As future work, We plan to look for an approach that combine timed Petri nets and colored Petri nets that we used in previous works [Elk00].

7 Bibliography

[And93] Anderson J. S., and Durney B., "Using Scenarios in Deficiency-driven Requirements Engineering", Requirements Engineering'93, IEEE Computer Society Press, pp. 134-141, 1993.

[Boo94] Booch G., Object Oriented Analysis and Design with Applications, Benjamin/Cummings Publishing Company Inc., Redwood City, CA, 1994.

[Dan97] Dano B., Briand H. and Barbier F. : An Approach based on the Concept of Use Case to Produce Dynamic Object-Oriented Specifications, In proceeding of the Third IEEE International Symposium on Requirements Engineering, pp.54-64, Annapolis, 1997.

[Del98] Delatour J., Paludetto M.: UML/PNO, a way to merge UML and Petri net objects for the analysis of real-time systems ". OO Technology and Real Time Systems Workshop (ECOOP'98), Bruxelles (Belgique), Lecture Notes in Computer Science, Springer Verlag, 1998.

[Elk98] Elkoutbi M. and Keller R. K. : Modeling Interactive Systems with Hierarchical Colored PetRi Nets, In Proc. of 1998 Adv. Simulation Technologies Conf., pp.432-437, Boston, 1998.

[Elk00] Elkoutbi M. and Keller R. K. : User Interface Prototyping based on UML Scenarios and High-level Petri Nets. In Application and Theory of Petri Nets 2000, Springer-Verlag LNCS 1825, pp.166-186, Aarhus, 2000.

[Hsi94] Hsia P., Samuel J., Gao J., Kung D., Toyoshima Y., and Chen C., "Formal approach to scenario analysis", IEEE Software, (11)2, pp. 33-41, 1994.

[Jac92] Jacobson I., Christerson M., Jonson P., and Overgaard G., Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1992.

[Jac99] Jacobson I., Booch G., and Rumbaugh J.: The Unified Software Development Process. Addison-Wesley, 1999.

[Ram80] Ramamoorthy C. V., Ho Gary S.: Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets. TSE 6(5): 440-449, 1980.

[Ram73] Ramchandani C. Analysis of asynchronous concurrent systems by timed Petri nets. Ph.D. MIT, 1973.

[Rum91] Rumbaugh J., Blaha M., Premerlani W., Eddy F., and Lorensen W., Object-oriented Modeling and Design, Prentice-Hall, Inc., 1991.

[Rum99] Rumbaugh J., Jacobson I., and Booch G., The Unified Modeling Language Reference Manual, Addison Wesley, Inc., 1999.

Salah A., Dssouli R. and Lapalme G.. Intégration de scénarios temps-réel en automates temporisés, Actes du Colloque Francophone sur l'Ingénierie des Protocoles (CFIP'2002), Montréal, Canada, 2002.