The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

UML EXTENSIONS FOR MODELING REAL-TIME AND EMBEDDED SYSTEMS

Sławomir SZOSTAK[1], Silva ROBAK[2], Roman STRYJSKI[1],
Bogdan FRANCZYK[1],

[1] Pedagogical University of Zielona Gora, Technical Institute,
ul. Wojska Polskiego 69, 65-762 Zielona Gora, POLAND, *S.Szostak@wsp.zgora.pl*,
Stryjski@post.pl
[2] Technical University of Zielona Gora, Institute of Computer Science and Management,
ul. Podgorna 50, 65-246 Zielona Gora, *Robak@pz.zgora.pl*

Abstract. *The process of modeling and developing of real-time and embedded systems should be supported by suitable methods and notations. In the paper we examine different approaches for customizing standard modeling language UML to model such systems in object-oriented analysis and design. We propose the use of UML standard lightweight extensibility mechanisms (stereotypes) without changing the UML metamodel. Our approach allows joining advantages of extended sequence diagrams and timing diagrams with UML and provides traceability of a concept throughout system development. The examples illustrate our approach. Applying lightweight UML extension mechanism allows existing standard UML modeling tools to be used without any adaptations.*

Key Words. *Real-time and embedded systems, UML extensibility mechanisms, extended sequence diagrams*

1. INTRODUCTION

The Unified Modeling Language (UML) adopted by OMG [8] as its standard modeling language has emerged as the software industry's dominant language. UML is a general-purpose graphical language for specifying, constructing, visualizing, and documenting workproducts that are modified, or used by software-intensive systems [1]. The UML needs to be extended for proposes of modeling real-time and embedded systems. It can be done either by using UML lightweight extensibility mechanisms (such as stereotypes, constraints and tagged values) or by heavyweight extension mechanisms - metaclasses. Metamodel level is a one layer of the UML's four-level model architecture based on metamodel architectural pattern [5]. The metamodeling offers significant advantages. It allows formal specification of all modeling concepts (together with their attributes, constraints and relationships), defines a base for unified exchange format and makes possible the extendibility of UML, i.e. instantiation of new metamodel classes as subclasses of the existing metamodel classes. Although changing the metamodel underlying the UML offers the highest degree of

flexibility, we have not taken it into consideration because the metamodel is not accessible or difficult to modification in existing UML modeling tools.

In the paper we present an approach for modeling real-time and embedded systems using UML. We present a concept for distinguishing model elements with stereotypes and then we examine known approaches for extending UML such as extended sequence diagrams for modeling real-time systems and also the use of timing diagrams.

2. EXTENDS OF STANDARD UML FOR REAL-TIME SYSTEMS

2.1. Stereotypes

Lightweight extension mechanisms are represented in UML metamodel as metamodel's classes named Stereotype, Constraint and TaggedValue. Stereotypes are a way of extending the basic metamodel to create a new model element as a subclassification of an existing model element. Stereotypes are used to mark, classify, or introduce new model elements in metamodel class hierarchy. Every model element may be marked with at most one stereotype, which is depicted in front of an element's name enclosed in double angle brackets, and/or represented graphical as an icon.

To model an element, which corresponds to a feature of real-time and embedded systems, we may introduce new stereotypes i.e. the objects such as a processor can be divided into the processors <<cisc>> and <<risc>> by using a stereotype, and thus give them the different features. The UML already predefines some stereotypes for classes, messages, objects, and etc [7]. The instance class containing the stereotype <<active>> is shown in Figure 1.

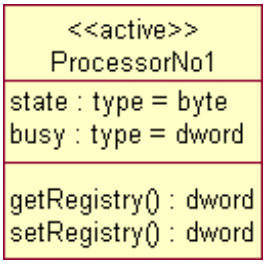

Fig. 1. Example class with stereotype <<active>>

Examining the real-time systems at building the stereotypes, it should take the characteristic features which derive from a given application domain into consideration. Suppose our system consists of the typical elements of industrial automation such as: processors, drivers, sensors, actuators, networking, monitoring, etc. For these and similar systems, i.e. embedded systems., safety-critical systems, the instance stereotypes can be distinguished in the UML for various elements. They are presented in Table 1-3.

Table 1. Stereotypes for nodes

UML Type	*Stereotype*	*About stereotype*
Node	<<processor>>	represents device that executes software
	<<other device>>	device that can not executes any software
	<<sensor>>	device that monitors course of external processes

<<actuator>>	Device that aktuates external process or other internal device
<<display>>	device that displays information for external actor (user)
<<knob>>	input device for external user
<<button>>	input device for external user
<<switch>>	input device for external user
<<watchdog>>	sensor that waits for fail-safe behaviour

Table 2. Stereotypes for messages (communications)

UML Type	*Stereotype*	*About stereotype*
message	<<synchronous>>	association realized as simple method call (directly)
	<<asynchronous-local>>	association that crosses a thread boundary and put the message in target thread's queue
	<<asynchronous-remote>>	association that crosses a processor boundary and put the message in target thread's queue without waiting for answer
	<<synchronous-remote>>	association that across a processor boundary and block sender until receiver returns answer
	<<periodic>>	message is sent periodically
	<<episodic>>	message is sent when event occurs
	<<epiperiodic>>	message is sent periodic and when event occurs

Table 3. Stereotypes for classes

UML Type	*Stereotype*	*About stereotype*
class	<<active>>	class is the root of an operating system thread

2.2. Scenarios

In each system some processes, which range the definite objects of this system, occur. Each of these processes consists of the elementary entities (i.e. external and internal calls, messages, interacts with actors, between objects etc.), whose chronological set composes a certain path or a branching tree through the system behavior. Such a path (or a branching tree) is called a scenario. Each scenario is based on a set of the objects and actors. The system behavior is composed of many completely independent and/or partly correlated paths. Only many such as scenarios produce a full image of the use-case system. Scenarios contain information about events both important and incidental for a system, but mostly scenarios are constructed basing on the most important elements. If scenarios differ only in the incidental elements, they are ignored. There are some various methods to describe a scenario: textual description, sequence diagrams and state diagrams [2]. The first method is not interesting because of its informality.

The state diagrams do not distinguish themselves anything specific for the real-time systems. Therefore, we study the extended sequence diagrams.

As we know, a sequence diagram shows the flow of messages between the objects of the system and the actors (Figure 2).

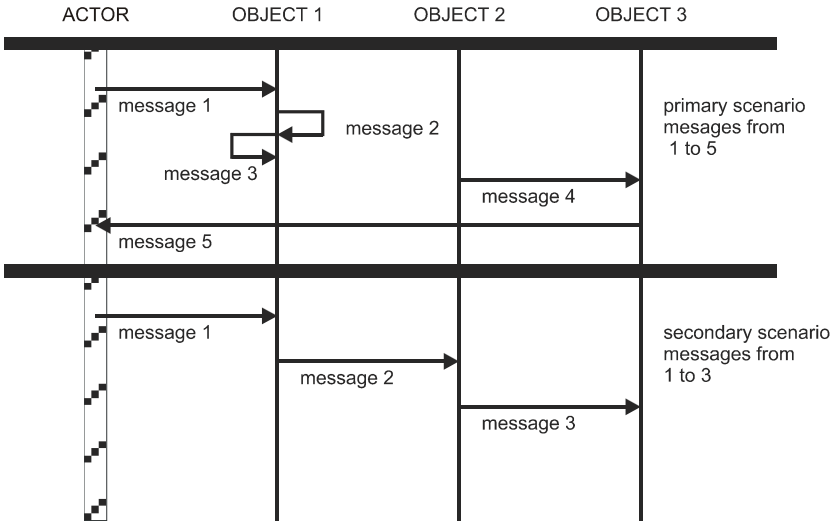

Fig. 2. Simple sequence diagram example

This diagram does not regard any following requirements for the real-time systems:

- execution time of event or message
- rise and fall time
- initiation and dwell time
- slack time
- deadline
- period
- leading and trailing jitter.

Therefore, some additional elements of a sequence diagram have been introduced for the real-time systems:

- timing marks: simple and conditioned
- state marks
- event mark.

Timing marks definite the duration of the time of a single event or message. This time is indicated between the start and the end of message (i.e. {< 20 ms}), or as the interval between the events (i.e. $\{t_1 - t_2 <= 10$ ms$\}$). It is called a simple timing mark. We may also indicate the duration of the time of a greater number of events (i.e. $\{t_5 - t_1 < 0.5$ s, but $t_3 - t_2 < 100$ ms$\}$).

Event marks represent the events that give rise to the message on the time line referred to the relevant object. The letters or symbols (shown on Figure 3 as indexed) on the time line written at the opposite ends of the message arrow, respectively indicate them.

State marks are to bridge a certain gap between the sequence and state diagrams. State diagrams do usually not depict the time dependencies between the states, and sequence diagrams do not show the present state of the system. State marks are the rounded rectangles placed on the time line off to the relevant object.

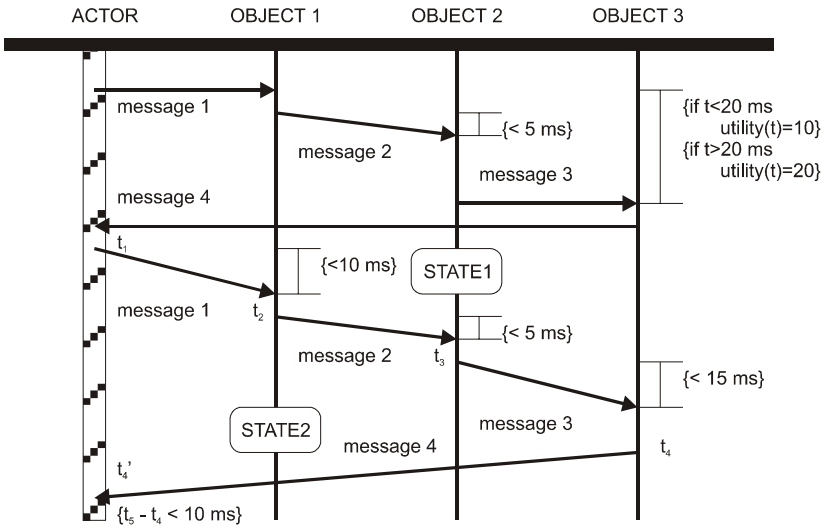

Fig. 3. Extended for real-time sequence diagram example

Timing diagrams are another way of representing a path of the system behavior. They have been known to electrical engineers for a long time (as well as to people who focus on programming the industrial controllers) as the diagrams being used in designing the electrical state machines (digital). They are, however, extended: the axis X depicts the time, but the axis Y can represent more than two states: on and off (or 1 and 0, H and L). Along the axis X there are some gaps, which separate the different states. If the system is in the defined state, a line (function) will be drawn in that state. On the axis Y along the state line there are special names of the states as well as the indicated events that give rise the relevant state (Figure 4).

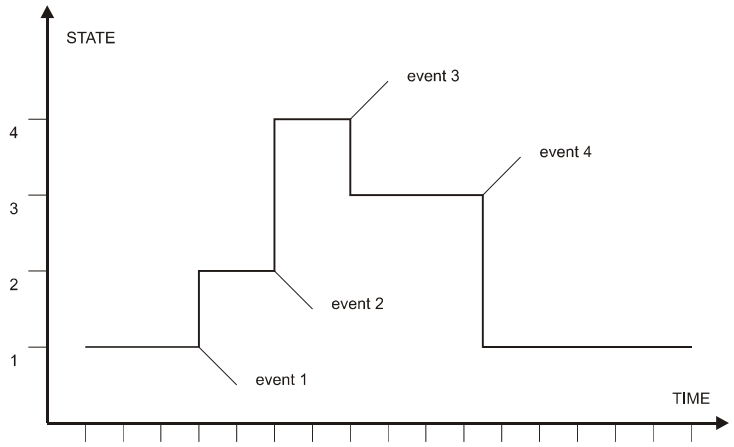

Fig. 4. Timing diagram for one scenario

This diagram depicts a development of events in time in a simplification, however. This considers only the timing of the particular states and enables it to show the only one scenario (for the only one object). It is possible to place more than one scenario in the timing diagrams, regarding the respective periods of the duration of the states (Figure 5 and 6).

3. SUMMARY

The approach described above allows combining the advantages of standard modeling with UML diagrams adopted for real-time and embedded systems. Traceability of a concept throughout system development is provided. Using only lightweight UML extension mechanisms (stereotypes) means, that existing standard UML modeling tools can be used without any extensions or adaptations.

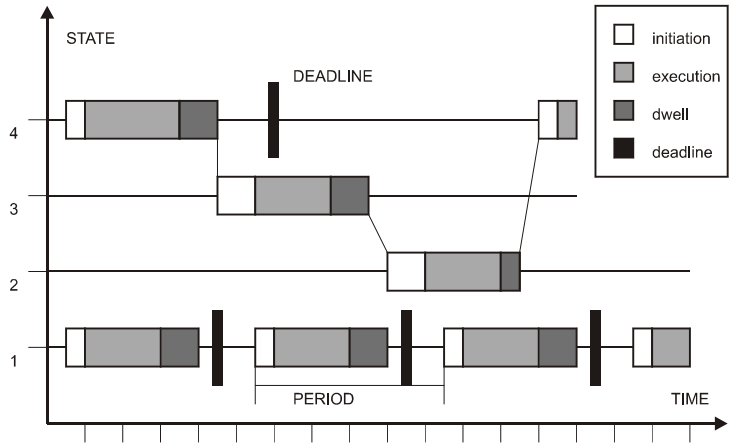

Fig. 5. Timing diagram for multiple scenarios

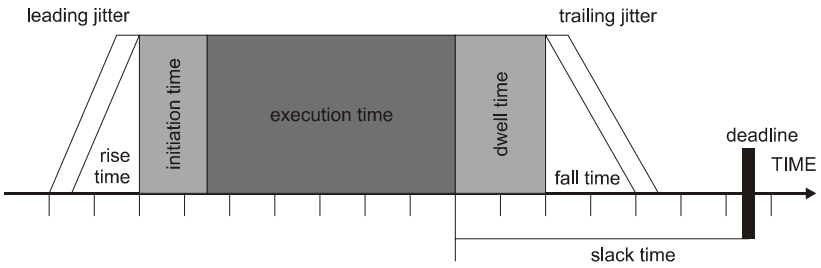

Fig. 6. Complex timing diagram for one event of scenario

In order to reach more comprehensive support for real-time and embedded systems modeling the next step is the integration of their features into the UML and specification of appropriate constructs to be defined as a special real-time and embedded systems UML profile.

REFERENCES

[1] G.Booch, J.Rumbaugh and I.Jacobson, The Unified Modeling Language User's Guide, *Addison Wesley*, 1999

[2] B.P.Douglass, "Doing Hard Time", *Addison-Wesley*, 1999

[3] B.P.Douglas, "The UML For Systems Engineering, *I-Logix whitepapers*, *www.i-logix.com*

[4] N.Hilary, "Bringing the Gap between Requirenments and Design With Use Cases and Scenarios". *I-Logix whitepapers*, *www.i-logix.com*

[5] C.Kobryn, "UML 2001: A standarization odyssey. Communication of the ACM", *Vol.42, No. 10, pp. 29-37*, 1999

[6] R.J.Muller, "Bazy Danych. Język UML w modelowaniu danych", *Mikom*, Luty 2000

[7] "OMG Unified Modeling Language Specification v 1.3", *Object Management Group*, March 2000

[8] OMG, Object Management Group, *http://www.omg.org*, 2000

[9] R.Rinat, "A Framework-Based Approach to Real-Time Development with UML", *I-Logix whitepapers*, Israel, June 2000