

New features in UML syntax and semantics

Michał Wolski¹, Andrzej Stasiak²

¹ Akademia Podlaska, Instytut Informatyki,
michal@iis.ap.siedlce.pl

² Wojskowa Akademia Techniczna, Instytut Teleinformatyki i Automatyki,
stasiak@ita.wat.edu.pl

Abstract. This paper is an overview of the most important new features introduced to version 2.0 of Unified Modeling Language. We denote the changes to existing diagram and present four new modeling techniques. We present the changes in definitions of syntax, semantics and pragmatics of behavioral and static models of systems designed with UML 2.0. Particular emphasis is put on the changes to sequence and activity diagrams and to component diagrams. The newly introduced diagrams, unknown in previous UML versions – interaction overview diagrams, timing diagrams, composite structure diagrams and package diagrams – are described with the most detail.

1 Introduction

UML (Unified Modeling Language) is a normalized language (supported by OMG – Object Management Group) used to create design models of IT systems. This language has been evolving for more than 10 years, and recently version 2.0 became the official new standard, providing new and enhanced features for behavioral and static system modeling. In UML 2.0 the language elements definitions have been specified with more details, two diagrams have been renamed. Four new diagrams were introduced – there were 9 diagrams in the previous version (UML 1.5), UML 2.0 has 13 diagrams.

2 UML 2.0 techniques – an overview

The main part of designed system's model is created using techniques known from previous UML versions. Two diagrams have been renamed, and the diagram set known from previous versions (1.x) has been complemented with package diagrams and composite structure diagrams, as well as interaction overview and timing diagrams used for behavioral modeling – the timing diagram is especially important in real time systems modeling. The class diagram below (fig. 1) presents

current (UML 2.0) diagram structure. Both the new and renamed diagrams have been marked.

The class diagram on figure 1 is put inside a frame not by a chance – all UML 2.0 diagrams have to have a frame and diagram's name and two letter abbreviation has to be put in upper left corner.

In UML 2.0 two diagrams used in behavioral modeling have been renamed. Communication diagram has been previously called Collaboration diagram, and the State Machine diagram has been previously named Statechart diagram. Neither those two diagrams, nor the Object and Use Case diagrams have been otherwise changed, so they won't be discussed any further.

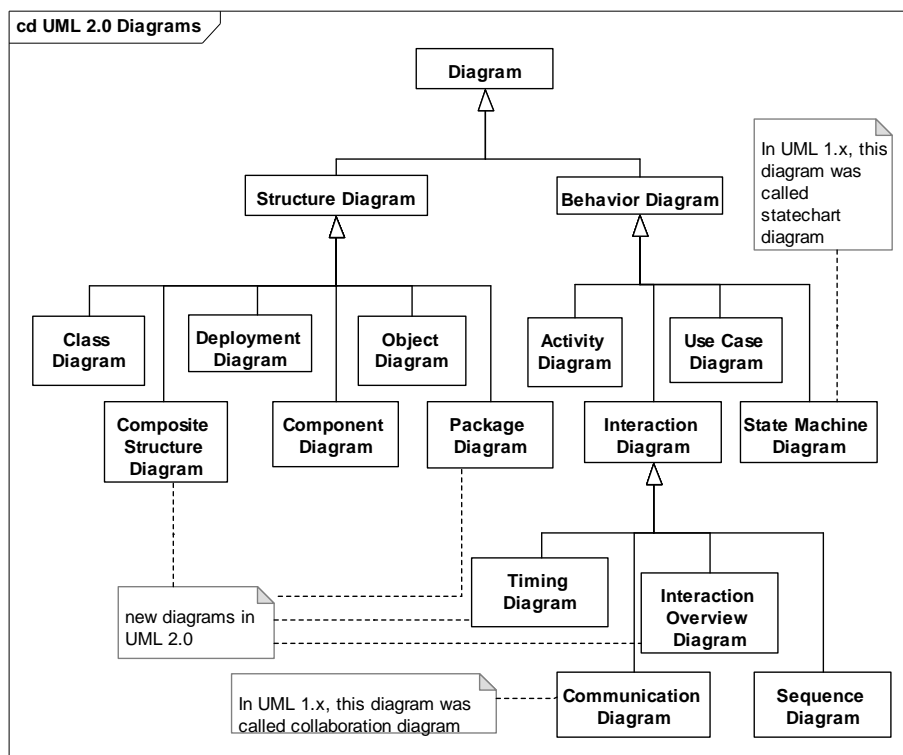


Figure 1. UML 2.0 Diagrams

3 Changes to previously known diagrams

UML 2.0 introduces more detail to many language elements known from previous versions. Those modifications are described in the following sections.

3.1 Class Diagrams

The changes in class diagrams introduced in UML 2.0 remove some ambiguities from notation.

First such element is the multiplicity of associations. UML 2.0 better details the multiplicities by adding (in curly braces) keywords representing type of the multiplicity [1]:

set - unordered, unique (duplicates not allowed) elements default;

bag - unordered, non-unique (duplicates allowed) elements;

ordered- ordered, unique elements ;

list (or *sequence*) ordered, non-unique elements;

Fig. 2 presents a sample class diagram using composition, which shows us that a single hotel has between 1 and 25 non-duplicate hotel rooms.

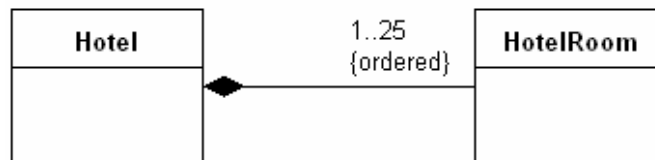


Figure 2. Multiplicity example

The same rules apply to specifying multiplicities of attributes and objects. Another change is a reduction in number of dependency relationship types. In UML 2.0 dependency relationship types <<local>>, <<parameter>> and <<global>> are not used any more [3].

3.2 Component Diagrams

Component diagrams in UML 2.0 were not changed much when compared to previous versions. The representation of some entities (pictograms) has been changed. The first important change is the modification of the main element of component diagrams – the component. This changes both the notation and role of a component. UML 2.0 dictates presenting components as classes with the <<component>> stereotype – optionally a class with a small component pictogram known from version 1.x may be used (fig 3).

UML 2.0 changes not only notation, but also the scope of component's usage. In previous UML versions a *subsystem* element has been used to model the physical system architecture. UML 2.0 abandons the *subsystem* element and uses component as an aggregate for subsystems on all levels of system design.

Elements known from UML 1.x such as subsystem realization and subsystem specification have been replaced by component stereotypes <<realization>> and <<specification>>. A new stereotype dedicated for large scale components [6] <<subsystem>> has been introduced.

Interface is another component diagram element with modified notation. UML 2.0 allows utilization of pictograms known from previous versions of UML, but also defines new notation that makes the interface utilization more precise (fig. 3).

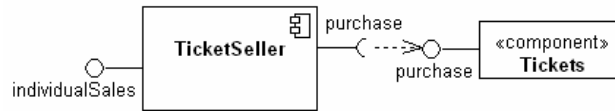


Figure 3. UML 2.0 interface notation

Interface notation defined in UML 2.0 (fig. 3) uses two interfaces called *connectors* used by a “TicketSeller” component. The first connector – “individualSales” is a Provided Interface, it indicates the fact, that TicketSeller provides individual sales operations to other components. The second connector is a Required Interface – an interface required for the component to function [4]. UML 2.0 allows for situations, in which a provided or required interface is not connected to its counterpart on a component diagram.

The component connection technique presented in the example (fig. 3) provides a noteworthy advantage – it allows the designer to trace components and interfaces required to realize and implement components providing essential system functions.

3.4 Deployment Diagrams

Deployment diagrams – like in UML 1.x – are used to present the configuration of deployed system’s nodes and installation locations of components. UML 2.0 introduces new elements. The first new element is *Deployment Specification*, responsible for description of parameters required to deploy component in a given node. Deployment specification is a tool used to parametrize relations between various software and hardware technologies. The next element added in UML 2.0 is *Device*, representing physical computer resources which may execute deployed artifacts [1]. Another new element is *Execution Environment*, representing selected platform type (i.e. operating system, database engine etc). The last modification is a change of connection stereotype from <<implement>> to <<manifest>>.

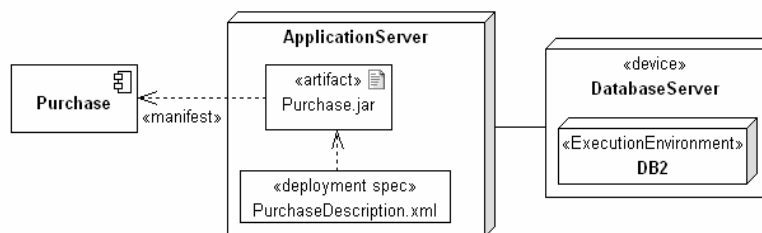


Figure 4. Deployment diagram example

The <<manifest>> dependency represents a physical materialization of one or more model elements by an artifact. The name change is caused by the fact, that the word “implement “ is used too often [5].

3.5 Activity Diagram

Activity diagrams in UML 2.0 have undergone serious changes. The changes apply to notation, semantics and pragmatics of those diagrams.

Table 1. Selected New Activity diagram elements


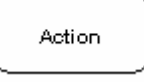

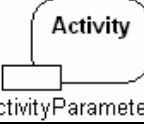



Symbol	Name	Description
	<i>Activity</i>	Parametrized system's behavior presented as ordered, subordinate elements, of which the Action is most important.
	<i>Action</i>	The basic action unit. An executable action node, which contains a transformation or a process of modeled system.
	<i>Flow Final Node</i>	A meta state representing a stop in activity execution before its completion.
	<i>Activity Parameter Node</i>	Objects existing at the beginning or end of an activity, that contain entry parameters and results.
	<i>Activity Partition</i>	A group of actions that have similar characteristics. Replaces swimlines from UML 1.x, may be vertical or horizontal.
	<i>Datastore</i>	Used to store persistent data.
	<i>Pin</i>	Represents flow of data to/from action. May be used to represent a flow of tokens.

Table 1 presents selected new elements used in activity diagram, their notation and description [5]. Other elements have not changed their semantics, and due to obvious reasons can't be described in detail.

Changes to activity diagrams were very extensive. Stages in activity diagrams in UML 1.x were called activities or activity states. In version 2.0 they are called actions and are not decomposed any further. An important modification in UML 2.0 is the fact, that activity diagrams are expressed using Petri-NET semantics. Another change is an unification in control and data flows. In UML 2.0 both flow types are represented by a continuous line, reserved for control flows only in previous versions. UML allows representation of data (objects) with technique known from UML 1.x (fig. 5), but also allows to represent flow of data streams as tokens. Action ports – Pins are used to represent such situations.

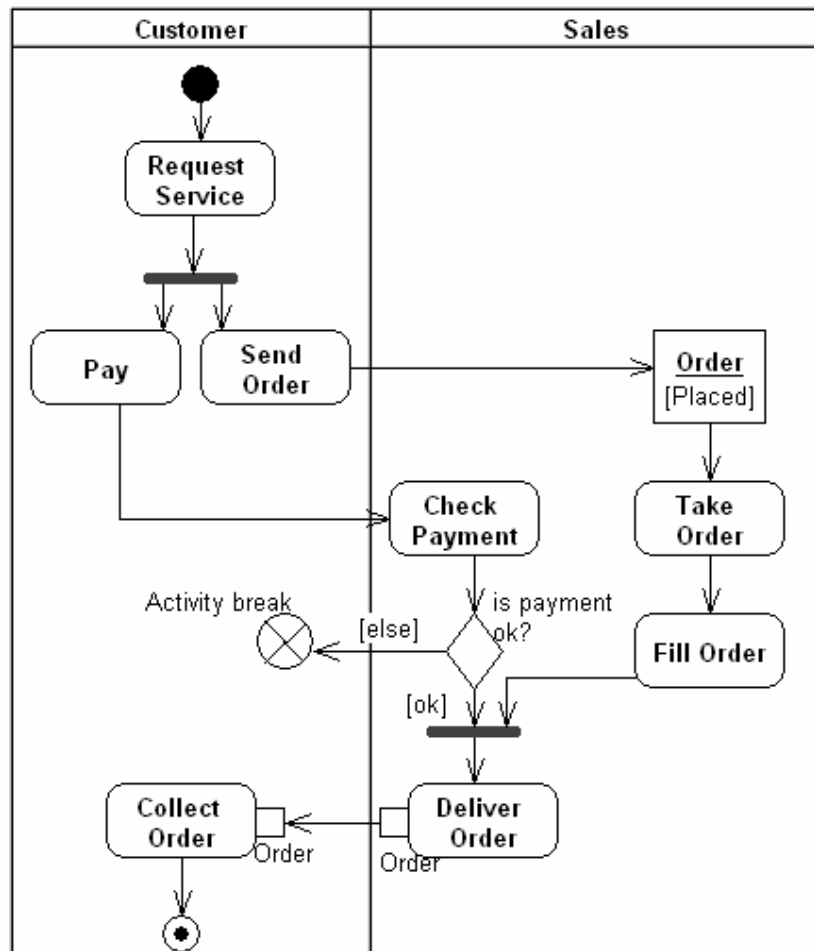


Figure 5. Activity diagram

3.6 Sequence Diagrams

Sequence diagrams in UML 2.0 have more detailed semantics when compared to previous versions. The most important new feature is the possibility of marking chosen, conceptually closed piece of diagram, which may be further refined. Those segments may be presented with interaction stereotypes and then connected with appropriate operators, thus allowing the designer to present variants and extensions to a scenario. Table 2 below contains chosen combined fragment stereotypes. A full specification of all stereotypes can be found in [5].

Table 2. Interaction operators (combined fragment)

Keyword	Description
alt	Divides a segment of interaction according to Boolean logic rules into two alternative scenarios. Each alternate path has to provide a guard which has to be fulfilled to execute given alternative.
assert	Specifies the only valid fragment to occur. Often enclosed within a consider or ignore operand.
break	Points to a segment of sequence diagram which will be executed if a condition is fulfilled. The fulfillment of a condition results in execution of a sequence of messages contained in the segment and then termination of the scenario. If the guard is not fulfilled, messages contained in the segment are omitted.
critical	Indicates a sequence that cannot be interrupted by other processing
ignore	Points to a segment of interaction containing messages that will be omitted, as their visibility does not change the system's behavior. The ignored messages will be listed after the ignore keyword.
loop	Interaction segment will be repeated specified number of times and has one subfragment with guard. The guard may have minimum and maximum count as well as a Boolean condition. If the guard is absent, it is treated as true and the loop depends on the repetition count
opt	Optional interaction segment, which is executed if a guard is fulfilled.
par	Represents parallel execution of message flow.
strict	Indicates the behaviors of the operands must be processed in strict sequence.

Another new feature is the possibility to point – through reference – to external interaction diagrams. In such case the connection between sequence diagram and an external diagram may be realized with a Diagram Gate or a fragment pointing to a reference that exists on all life lines. UML 2.0 allows also for messages, which do not have a known sender or receiver (fig. 6).

UML 2.0 also allows to abandon underlining names of the objects taking part in an interaction.

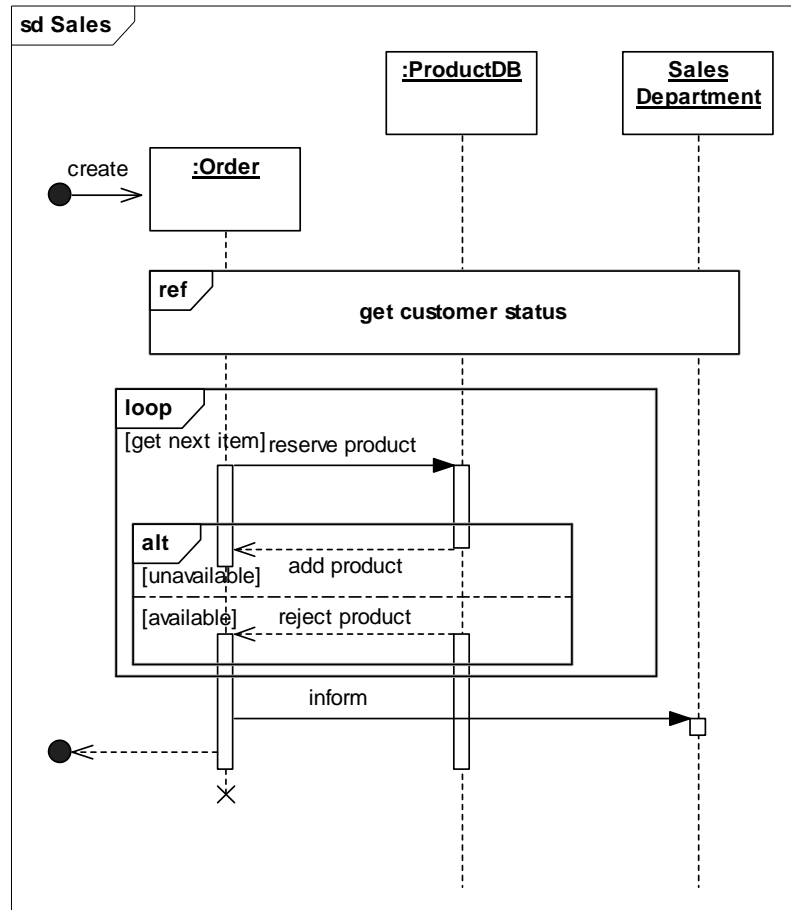


Figure 6. Sequence diagram

4 New diagrams in UML 2.0

The previous part of this paper described changes made to diagrams known from previous UML versions. The rest of this paper discusses new modeling techniques.

4.1 Package Diagram

Package diagram has been used in previous UML versions, in version 2.0 it has been approved as a standard. Notation and types of dependencies between the packages have not been changed. A new feature in UML 2.0 is a view of embedded package contents as a specification of its elements (fig 7). This mechanism is called package merge and is not intended for use by the ordinary modeler because this technique is destined primarily for metamodel builders forced to reuse the same model for several different, divergent purposes.[1]

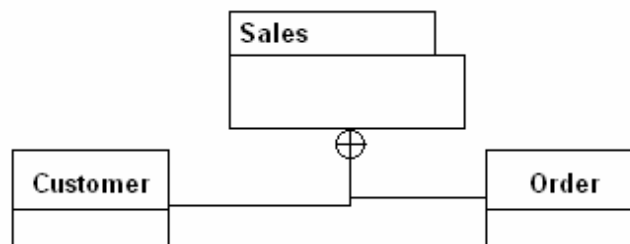


Figure 7. Merging the contents of package

4.2 Composite Structure Diagram

Composite Structure Diagram or *Internal Structure Diagram*[1] is a new feature in UML 2.0, and is used to describe internal structure of classifiers (e.g. classes, components, nodes, use cases) allowing for interaction points of a structural classifier with other system elements.

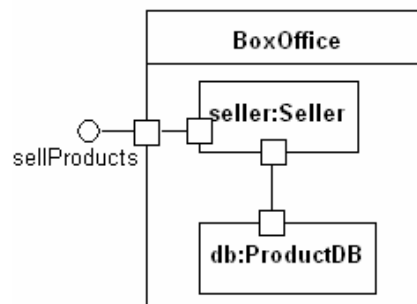


Figure 8. Composite structure diagram example

Composite structure diagrams allow the designer to model atypical cooperation between elements of the system architecture, which extraordinary specific was unknown during initial design phase. Structure diagrams present a decomposition of classifier parts. Those parts may be further decomposed. There is also a possibility of encapsulation of a structural class and connecting its part via a port with an external interface (fig. 8).

Another specific type of composite structure diagrams exists – one that represents relations between cooperating objects. It is described in detail in [1] and [5].

4.3 Interaction Overview Diagram

Interaction Overview Diagram is a new element of UML. This diagram is a specialized form of activity diagram and is used to present dependencies and message flows between interactions.

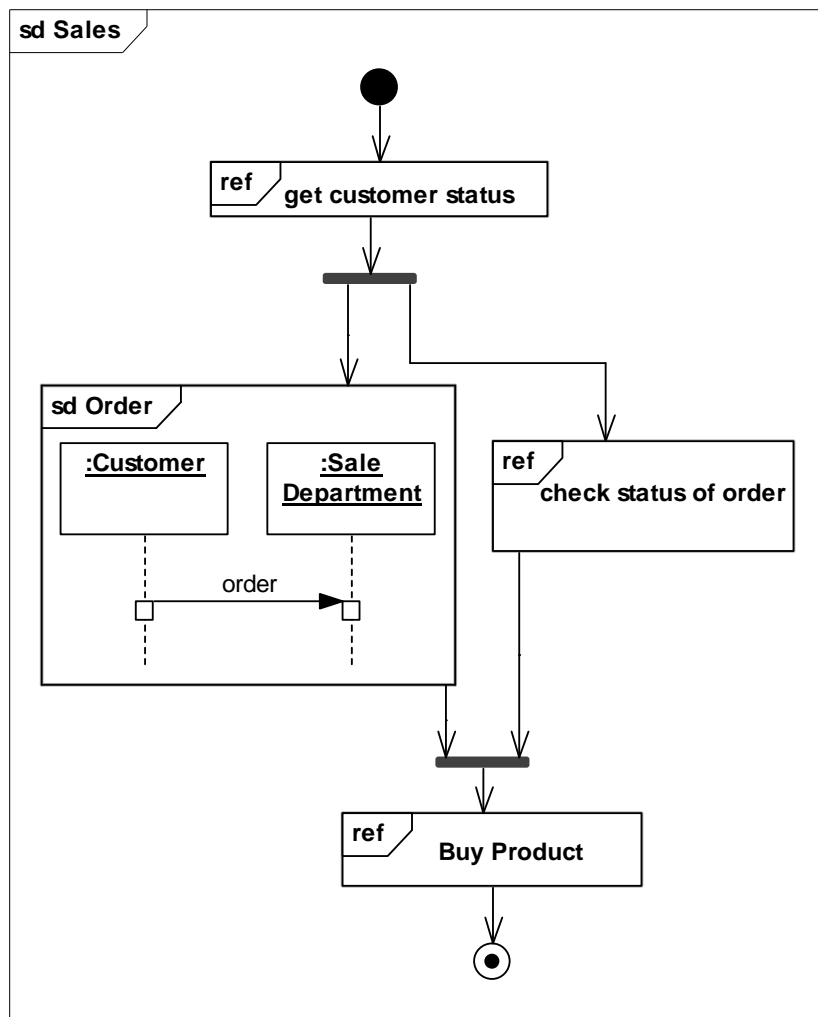


Figure 9. Interaction overview diagram – example

An interaction overview diagram uses the semantics and notation of activity diagrams, with one exception – instead of activities and actions nodes, rectangular elements symbolizing interaction diagrams or interaction segments are used.

During interaction overview diagram creation two types of interaction elements may be used. Rectangular elements with interaction diagram name and ref keyword in upper left corner are the first type – they represent references to those diagrams. The second type are sequence or communication diagrams embedded directly in the interaction overview diagram (fig. 9).

4.4 Timing Diagrams

Timing Diagrams are a new feature in UML 2.0. They are used to present the object's behavior in an environment with severe timing constraints [5]. Obviously the diagrams of this type will be most useful in real time system modeling. Timing diagrams are perfect for presenting the changes in state of objects and their influence over other objects. An additional advantage presented by timing diagrams is the possibility of time constraint modeling. An example of timing diagram with the description of its elements is presented on fig. 10.

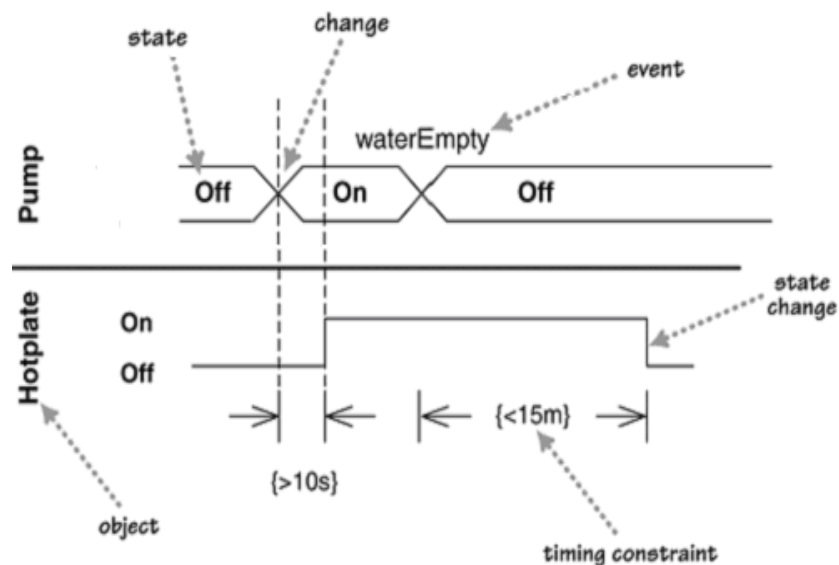


Figure 10. Example of timing diagram

5. Conclusions

All living languages evolve to allow its users to create statements or descriptions that adequately present past, present and new (previously unknown)

reality. The main IT providers organized in OMG reached a conclusion that after 7 years that passed since UML 1.1 has been made official, it is time to introduce essential changes leading to approval of its' new generation – UML 2.0.

The conclusions from analysis of the changes in notation and semantics of UML presented in this paper are obvious. The creators of UML are abandoning model simplicity in order to make the models more precise and formal. The future will tell, if this direction is the correct one.

References

1. Booch G., Rumbaugh J., Jacobson I.: *The Unified Modeling Language Reference Manual*, Second Edition, Addison Wesley, 2005.
2. Booch G., Rumbaugh J., Jacobson I.: *The Unified Modeling Language*, User Guide, Addison Wesley, 1998.
3. Fowler M.: *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Third Edition, Addison Wesley, 2003.
4. Kendall S.: *Fast Track UML 2.0*, Apress, 2004.
5. Object Management Group: *Unified Modeling Language: Superstructure*, Version 2.0, ptc/03-07-06.
6. Rational Unified Process ver. 2003.06.13.
7. Rosenberg D., Scott K.: *Use Case Driven Object Modeling with UML*, Addison Wesley, (1999).
8. Stasiak A. Wolski M.: *Zintegrowane środowisko wytwarzania aplikacji web'owych dla systemów mobilnych na platformie .NET*, XI Konferencja SCR'04, Ustroń 13-16 września 2004 r., 243-252.
9. Quatrani T.: *Visual Modeling with Rational Rose 2000 and UML*, Addison Wesley (2000).