BULETINUL Universităţii Petrol – Gaze din Ploieşti	Vol. LIX No. 3/2007	27 - 54	Seria Ştiinţe Economice

Cycle: UML FOR MANAGERS (VII) Interaction Diagrams

Liviu Dumitraşcu*, Gabriel Irinel Marcu**

*Petroleum-Gas University of Ploieşti, Bd. Bucureşti 39, Ploieşti
e-mail: ldumitrascu@upg-ploiesti.ro

** Halliburton Energy Sevices Romania S.R.L., Central Businesspark 133, Calea Şerban Vodă, 2nd Floor, Sector 4, Bucharest, România
e-mail: Irinel-Gabriel.Marcu@Halliburton.com

Abstract

Interaction diagrams describe the way in which objects communicate within a system, in order to achieve a certain function. This paper presents basic and advanced concepts of the dynamic interaction diagrams (UML 2): the sequence diagram, the communication diagram, the overview interaction diagram and the time diagram.

Key words*: UML diagrams, sequence diagram, communication diagram, time diagram*

Examples of Interaction Diagrams

Interaction Diagrams – A Bridge Between the Case-of-Use Diagram and the Class Diagrams

Interactions are numerous and various. In order to be able to express them we need an enriched vocabulary. UML proposes several types of interaction diagrams: the sequence diagram, the communication diagram, the global interaction diagram and the time diagram.

UML introduced the interaction diagrams in order to allow a better description of communication means between objects but not the manipulation of the involved data in this transaction. Interaction diagrams focus on the specific messages that are changed through objects and on the way these messages offer a better functionality to the system.

Figure 1 presents an example of an interaction diagram for an International Scientific Conference management application.

Class instances can appear on a sequence diagram (see figure 2) or on a communication diagram (see figure 3). The communication diagrams show the message flow between objects in an Object Oriented application and also imply the basic associations (relationships) between classes. The rectangles represent the various objects involved that make up the application. The lines between the classes represent the relationships (associations, composition, dependencies, or inheritance) between them.

In the example presented in figure 3, class instances send messages: *takeExam* and *answerQuestions*. In practical terms, sending messages triggers off the call of these two methods (*takeExam* and *answeQuestions*), the effect being consulting or changing group or student elements.

Figure 1

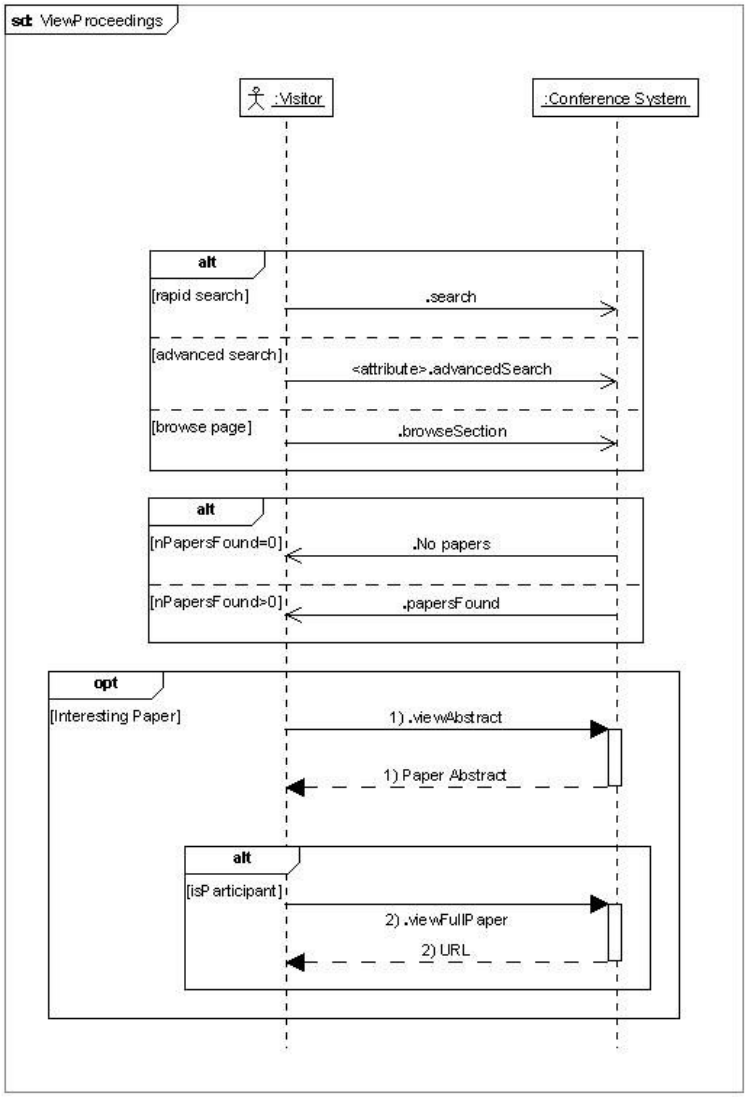


Figure 2

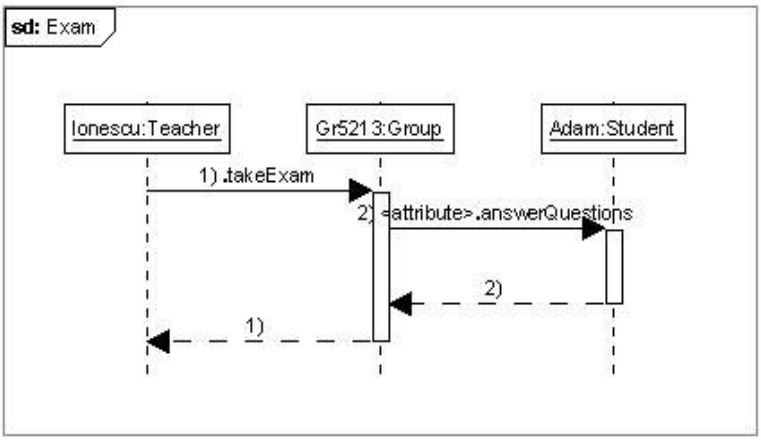
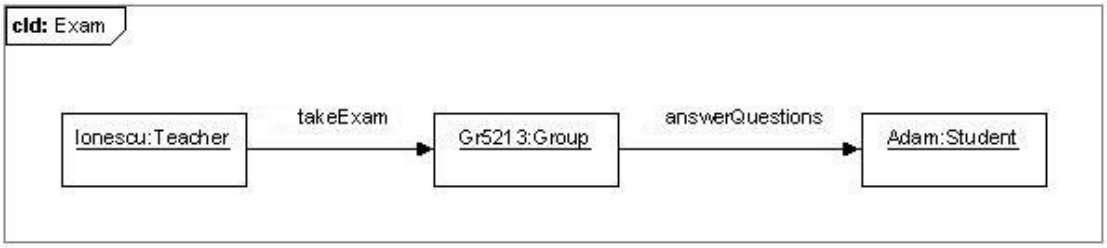


Figure 3



Remarks:

- The sequence diagram and the communication diagram represent the same interaction but their approach is different.
- A sequence diagram takes into account the temporal sequence of messages, vertically.
- In a communication diagram, the messages vector or the link between the lifelines do not appear.
- A lifeline represents a unique participant in an interaction.
- The sequence diagram and the communication diagram represent the interactions between object lifelines. A sequence diagram describes actions temporally, more precisely the temporal sequence of the changed messages between lifelines, while a communication diagram describes a spatial representation of life lines.

The Sequence Diagram

Notation

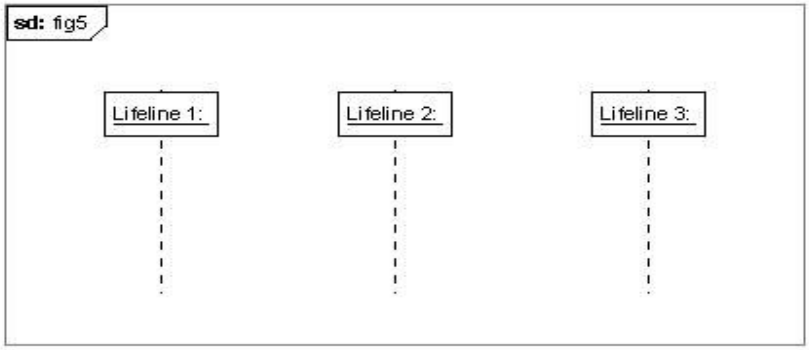
The sequence diagram describes the vertical sequence of messages that are changed between more lifelines (communication partners).

A sequence diagram is represent by a rectangle that contains in the top left corner, a pentagon in which the keyword (sd) (sequence diagram) that is required by the name of the interaction (figure 4).

Figure 4

The list of the lifelines that appear in the sequence diagram may be added to the name of the interaction, using the keyword lifelines (figure 5).

Figure 5



In a sequence diagram the interaction attributes may also be added (the interaction syntax is the same with that of class attributes) at the top of the rectangle containing the diagram.

A lifeline is represented by a rectangular (see figure 6) on which a vertical line is attached. The rectangle contains an identifier whose syntax is presented in figure 7.

Figure 6

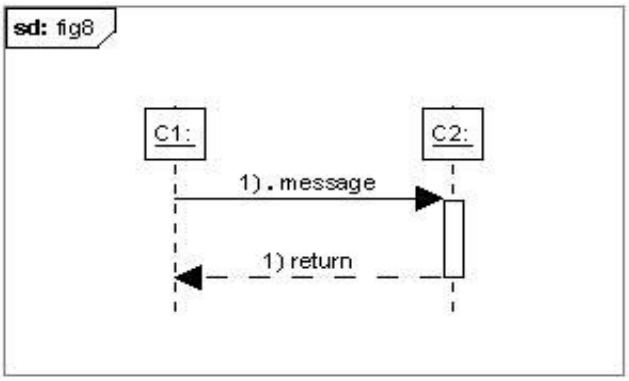
[<LifelineName>['['selector']']]
<ClassName> ref [decomposition]

Figure 7

Syntactic Element	*Comments*
lifelinename	*It is the name of the instance implied in the interaction*
Selector	*It allows the identification of an object from n objects (example object [2])*
Classname	*The name of the class*
decomposing	*It allows the reference of another interaction diagram. In this case we need to maintain the keyword ref.*

Figure 8 presents the significance of the syntactic elements of this notation.

Figure 8



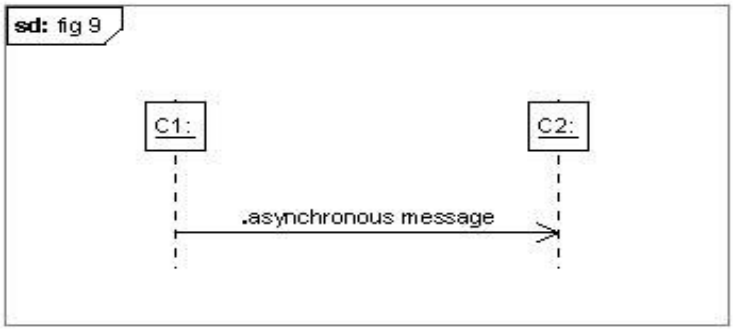
The interaction diagrams focus on the communication between lifelines.

The interaction between two lifelines (communication partners) can be produced through messages. There are more types of messages; the most important ones are the synchronous and the asynchronous messages.

In case of a synchronous message, the sender is stuck until the moment he gets an answer from the receiver. Synchronous messages are also known as operation calls, but they can also be signals.

A synchronous message is represented by an arrow with a full head pointing towards the receiver of the message. This message can be followed by an answer represented by an arrow with an open pointed head.

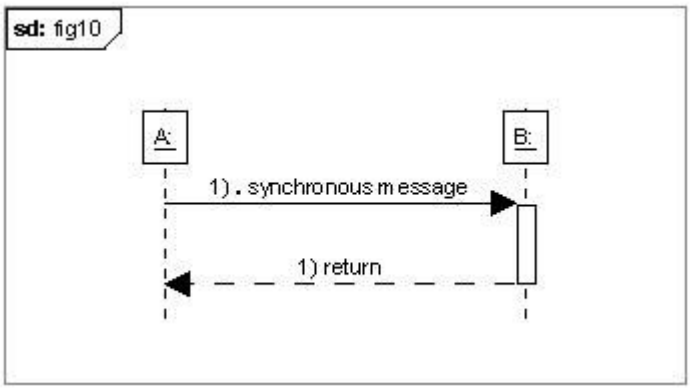
Figure 9



In the case of an asynchronous message, the sender does not expect his message to be received, but he can go on in parallel with his own received messages. Asynchronous messages appear as signals.

An asynchronous message is represented by an arrow with an open head (figure 10)

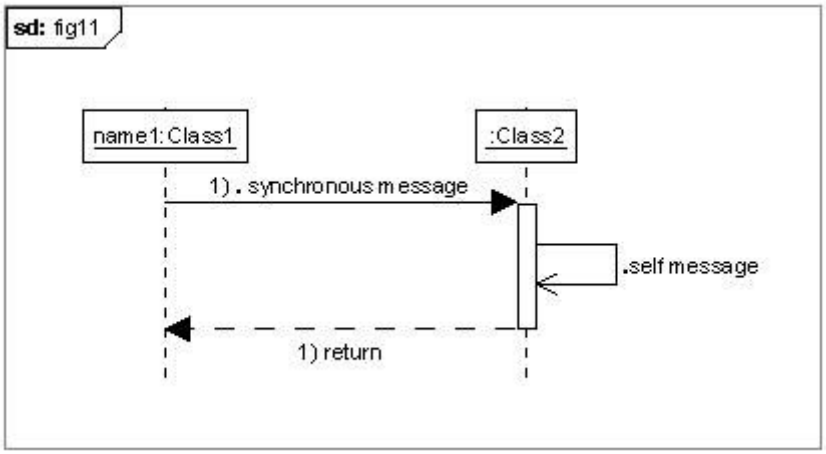
Figure 10



Each parallel lifeline, an active bar that shows the amount of time the communication partner is active. In fact, this corresponds to the time the respective operation is active.

The activation lines, represented by a grey or transparent line (see figure 11), are optional; they are very useful to classify object behaviour.

Figure 11

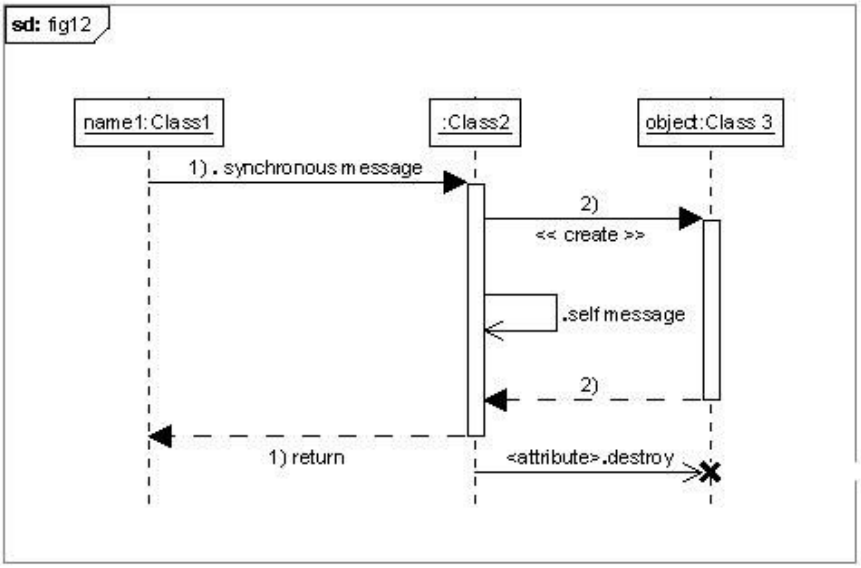


The imbrication of an activity period is represented by a loop that can be closed in the active bar. The imbrication is due either to recursivity or to an internal function call, or another object call.

The imbrication is represented through an active bar that can be closed by itself, and it is half symbolised inside the main active bar, towards the right (figure 12).

The majority of objects appearing in a sequence diagram appear all over the duration of interactions, being aligned at the top of the diagram. However there are also objects that are created while interactions appear, whose lifeline starts when the message “create” is received (figure 12). The creation of an object is materialised through an arrow pointing to the lifeline. The objects may be also destroyed during an interaction. In this case, their lifeline ends the moment the message “destroy” is received (figure 12).

Figure 12



The destruction of an object is materialised through a cross (X) that marks the end of a lifeline.

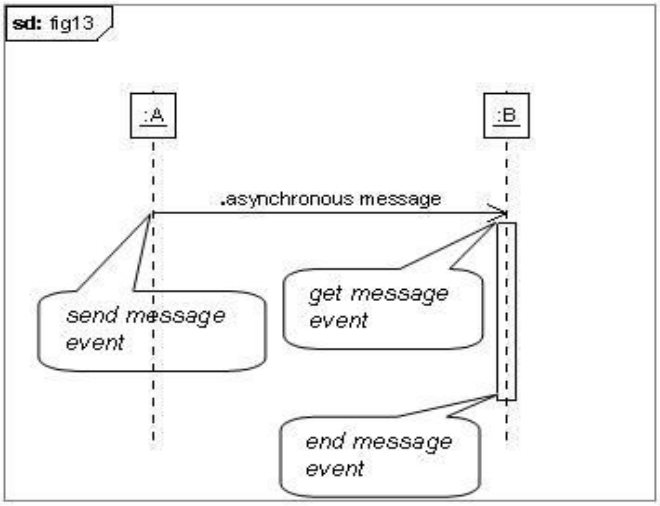
Messages and Events

Invoking an operation or sending a message may start a reaction to the receiver. The most common reaction is the execution of a method (a method is the implementation of an operation).

UML allows us the class separation of sending a message and receiving it as well as the beginning of the execution and its final reaction. The events are used to mark each stage. Sending a message is checked through two events: one used for sending messages, the other for receiving them.

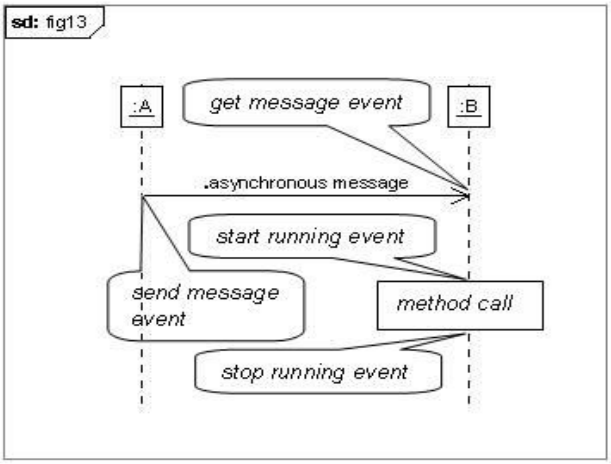
Figure 13 describes a sequence diagram in which an asynchronous message leads to the execution of a method at the receiver's end.

Figure 13



Remark. Specifying the execution of a reaction is done through a white or grey rectangle placed on the lifeline. Another notation may be a rectangle that contains a label (figure 14).

Figure 14



The Syntax of Messages

UML allows one to model each type of system. However, quite often, the implementations are done through the program languages. In the majority of the cases, receiving a message is

followed by the execution of a method that belongs to a class. This method may receive arguments (parameters), and the syntax of the messages allows one to send these arguments.

The syntax of messages and arguments is presented in figures 15 and 16.

Figure 15

<SignalOrOperationName>['('[<parameter>[,...]]')']

Figure 16

[<parameterName>'=']<parameterValue>

Implicitly, the sending parameters cannot be changed by the receiver (arguments "input"). In order to specify that parameters could be changed (arguments "input" or "input/ output"), one must obey the syntax presented in figure 17.

Figure 17

<InOutParameterName>[':' <parameterValue>]

When a message must send only the value of arguments, the sign "_" can be used in the place and position of an argument in order to mention: undefined value. The sign * can be used in the place and position of the complete message. It signifies the fact that any type of message is accepted.

Figure 18 displays more examples of messages.

Figure 18

1. message1(1,"Happy Birthday","-",1947)

⇔

message1(p1=10,p2="Happy Birthday",p3="-",p4=1947

2. y=f(x=0)

3. y=f(x):0

A message1 (1, "HappyBirthday!", _, 1947) is equivalent to a message (p1=10, p2="HappyBirthday!", p3=1947). The message *y=f(x=0)* is a reply to a message f(x=0); its return value is attributed to y. The message y=f(x):0 is a reply to a message f(x); its return value is attributed to y.

The receiver of a message may send a reply via return message, whose syntax is presented in figure 19.

Figure 19

[<Attribute>'='] message [':' <returnValue>]

Figure 20 illustrates the diagram of cases in which a simplified automatic distribution system is used for more products (SDAP). Figure 20 illustrates the sequence diagram presenting the used cases, obeying the global sequence based on buying a product: choice, payment, and then receiving the product.

Figure 20

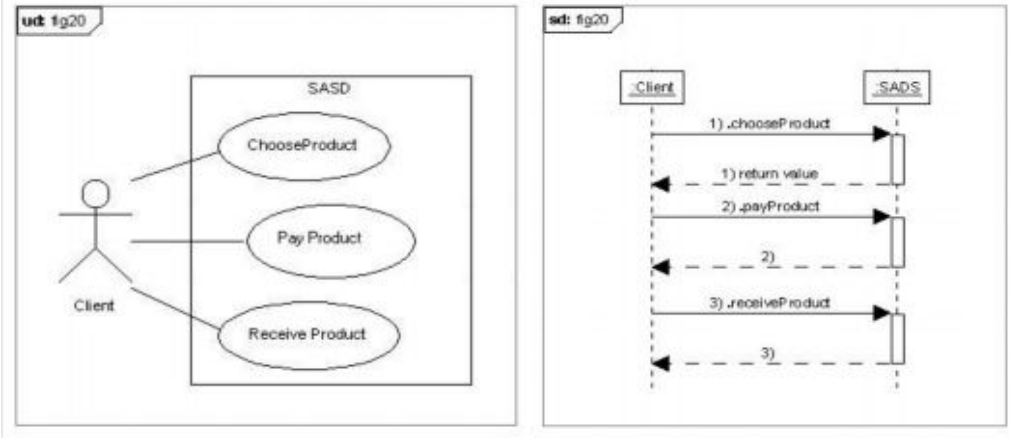


Figure 21 presents the sequence diagram for the calculus of the length of a circle with a given radium, in two sequences: a) Pi is calculated with four decimals, using a message SMSM; b) Pi is calculated with 400 decimals, using an asynchronous message.

Figure 21a

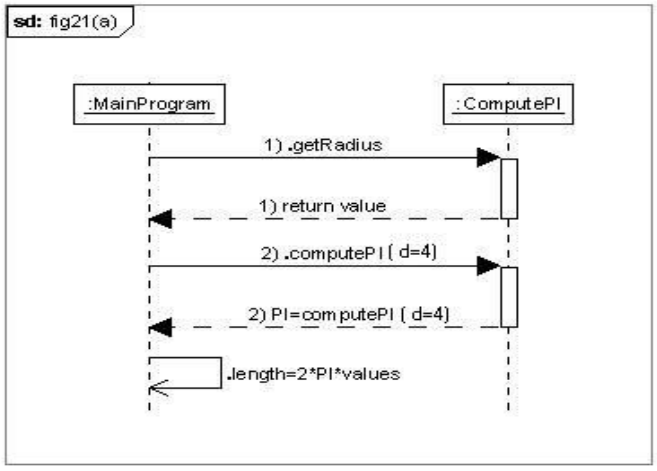
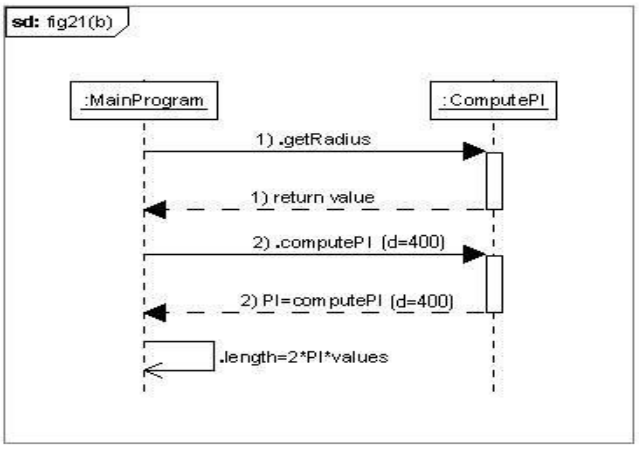


Figure 21b



Combined Interaction Fragments (UML 2)

In UML 1.4. and their previous versions, it was not possible to compose interactions, neither was it possible to integrate an interaction into another one. These incovenients created difficult problems as it was not possible to specify and reuse interactions for complex situations.

These drawbacks were completely removed in UML 2, which bears the concept of interaction fragment. Combining fragments allows the reconstruction of the system complexity. The term used in UML is "combined fragments".

A combined fragment is represented in the same way an interaction is represented, by a rectangle whose upper left corner contains a pentagon. Inside the pentagon the type of combination is formed. It is known as an "interaction operator".

UML 2 allows, by means of combined fragments, a more precise description of the control structures (selection, iteration etc.)

Each combined fragment is made up of an interaction operator and one or more interaction fragments, which represent the ones who operate interaction. An interaction operator describes the way in which the ones who operate interactions must be interpreted.

The ones who operate interactions are separated by a pointed line. The conditions in which they are chosen are expressed by logical Boolean expressions that appear between square brackets.

The Interaction Operators (UML2)

The interaction operators, grouped on functions are presented in table 1.

Table 1

Denomination of the interaction operator	*Function*
Alt, opt, break	*Decision*
Loop	*Iteration*
Par, critical	*It controls parallel message sending*
Ignore, consider, assert, neg	*It controls message sending*
Strict	*It fixes the order of message sending*

Figure 22 describes the interaction operators UML2, and it also specifies the function and its main uses. Follow the presented examples.

Figure 22

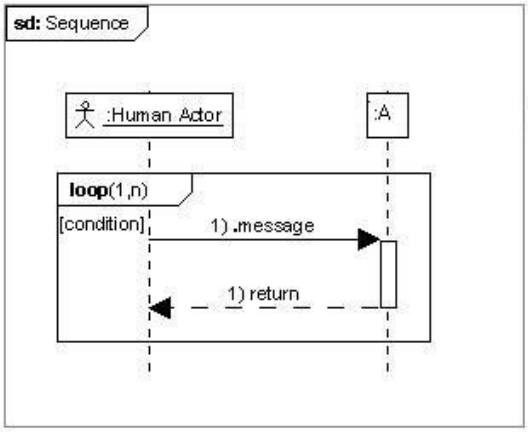
Interaction operator	**Function**	**Use**	**Example**
loop	Iteration operator (loop)	It defines an interaction fragment that is done several times. The logic condition explains the iteration.	

Figure 22. (cont.)

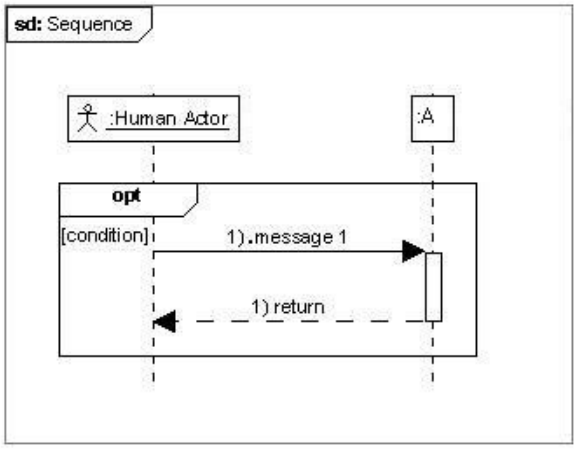
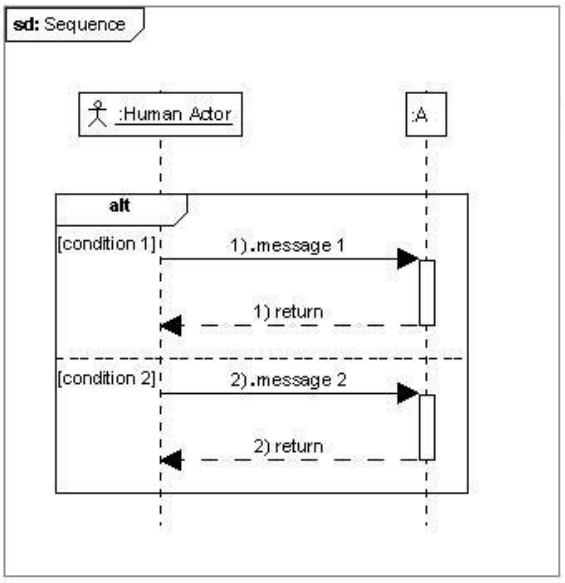
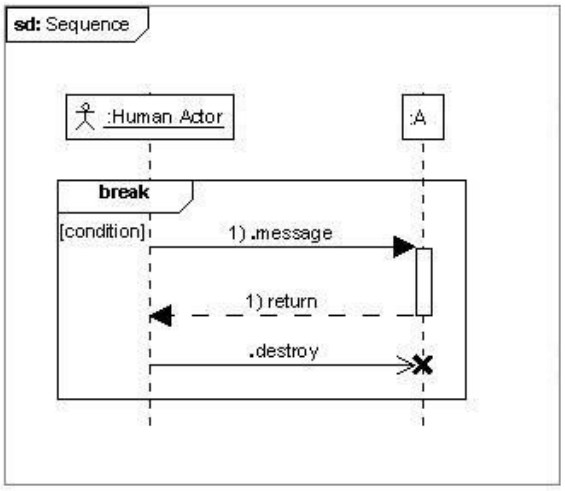
opt	Precision (selection) operator with one alternative	It defines an optional interaction fragment.	
alt	Decision (selection) operator with two or more alternatives	It defines alternative interaction fragments. It includes the "else" type logic condition	
break	Decision operator	It defines an interaction background corresponding to an exception. It is similar to the instruction: logic condition {…; return;}	

Figure 22. (cont.)

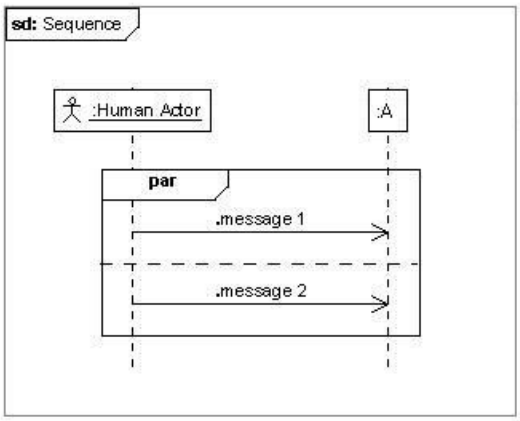
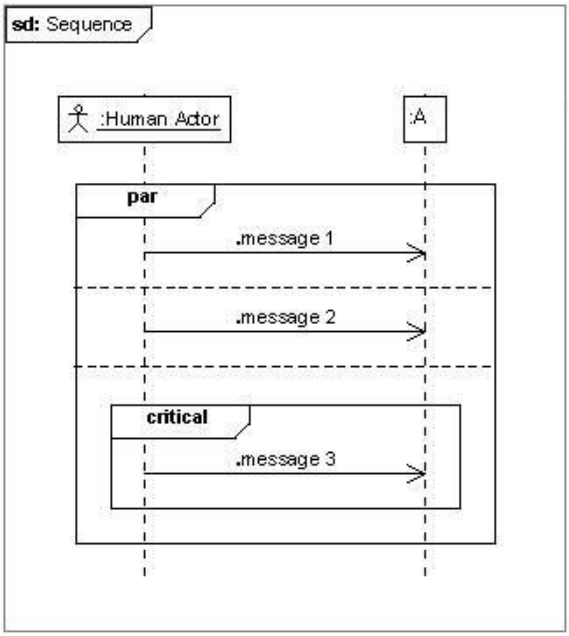
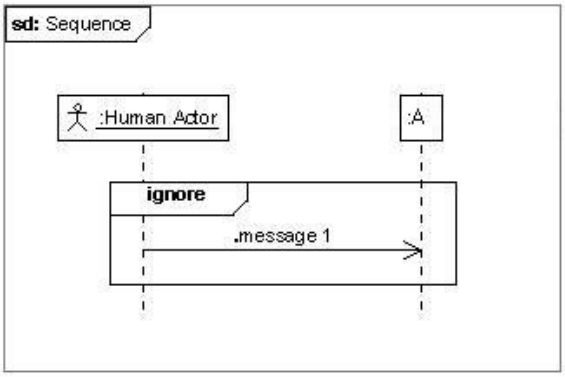
par	Operator that controls parallel message sending	It allows parallel message sending. The operations of the "alt" operator take place in parallel.	
Critical	Operator that controls parallel message sending	It defines a critical sequence – a part of an interaction in which processing is atomic.	
ignore	It controls message sending	It defines the amount of messages that can be ignored.	

Figure 22. (cont.)

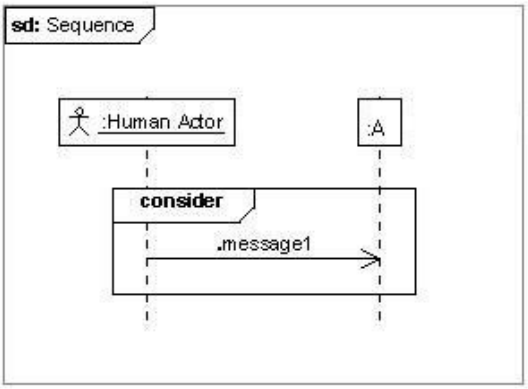
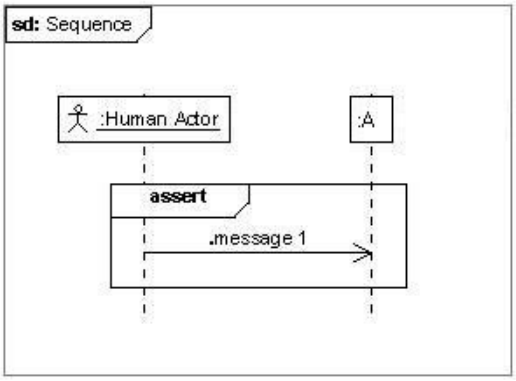
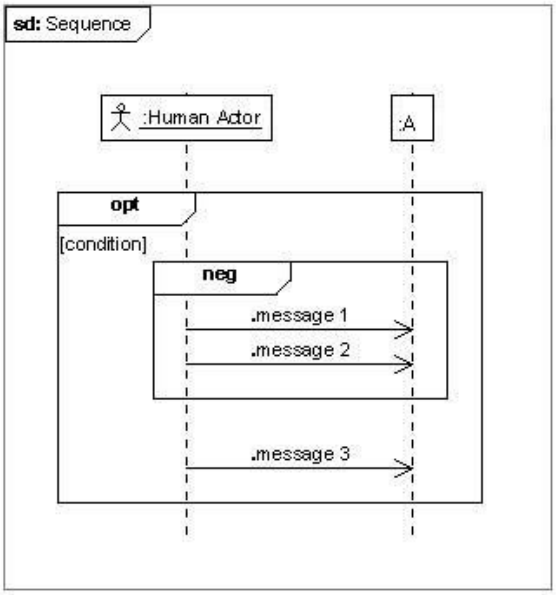
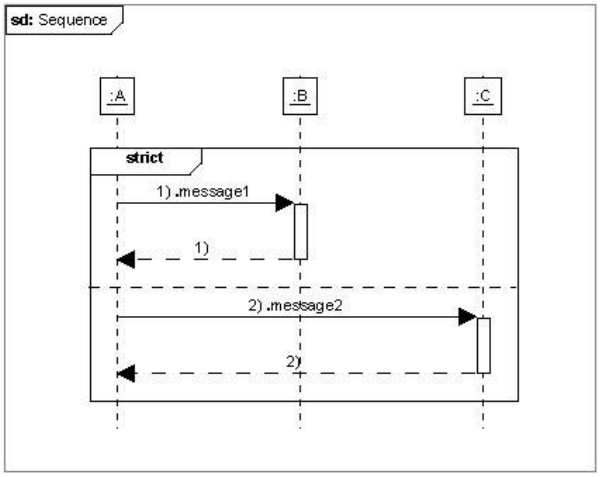
consider	It controls message sending	It defines the amount of messages that must be taken into account.	
assert	It controls message sending	It defines the amount of messages that represent the only way in which they can be processed. It also establishes the interaction conditions that must be fully obeyed.	
Neg	It controls message sending	It makes invalid the amount of messages that it is applied to.	

Figure 22. (cont.)

strict	It fixes the order of message sending (strict sequencing)	It imposes a certain order of message sending to be obeyed (top down order).	

Remark. Most of the times an interaction is too complex to be described in a single sequence diagram. UML allows decomposing a lifeline in more diagrams. A sequence diagram can be referred to with the "ref" operator, that indicates reusing an interaction fragment (see figures 23 and 24).

Figure 23

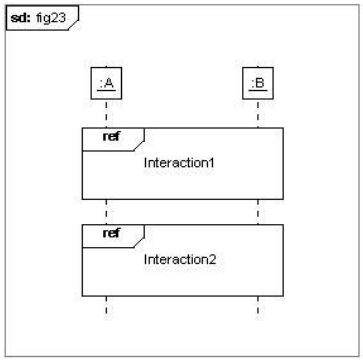
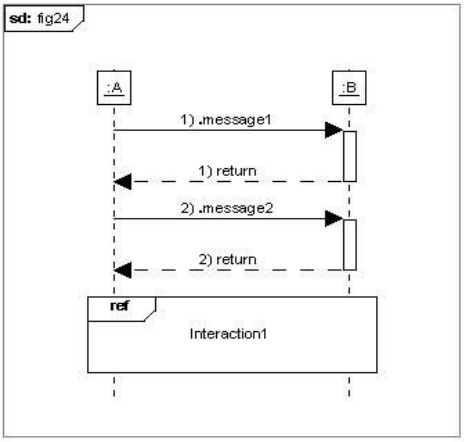


Figure 24



More on the Loop Operator

A loop is done with the loop operator followed by the parameters minint and maxint and a test condition (figure 25).

The content is done by <minint> and before the hypothetical test condition is tested, the condition is placed between square brackets, on the lifeline. As long as the result of the condition assessing is TRUE, the loop can be done <maxint>. Times at most (minint is whole number bigger or equal to zero), and maxint is a whole number superior or equal to minint.

Loop (value) is equivalent to loop (value, value).

Loop is equivalent to loop (0,*), where * means “unlimited”.

Figure 25

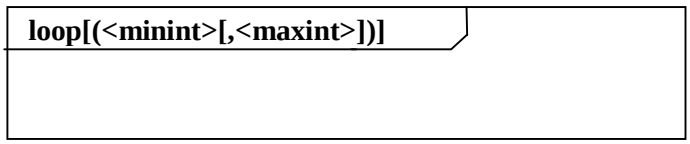


Figure 26 presents an interaction fragment that the loop operator uses, in order to calculate the sum of elements of a vector (see the class diagram in figure 27)

Figure 26

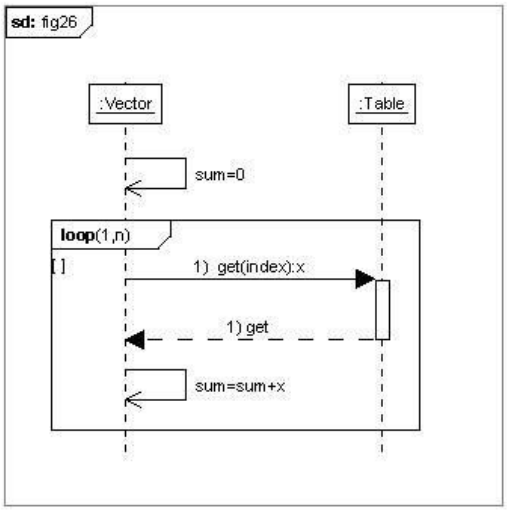
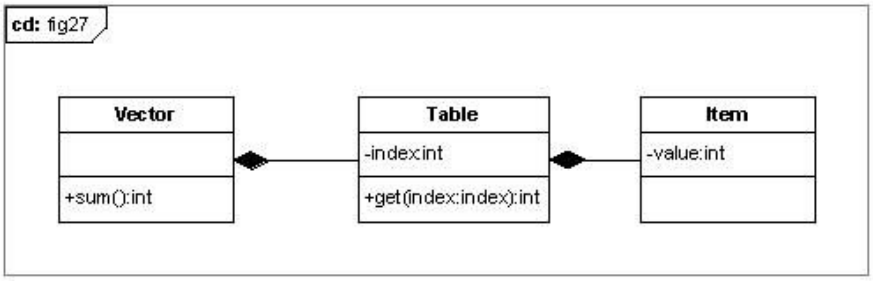


Figure 27



Remark: The method get(index) of the class Table returns the value of the element whose index is transmitted as a parameter.

Figure 28 presents an interaction fragment that the loop operator uses in order to calculate the sum of elements of a matrix (number of lines * by number of columns). (see the class diagram in figure 29)

Figure 28

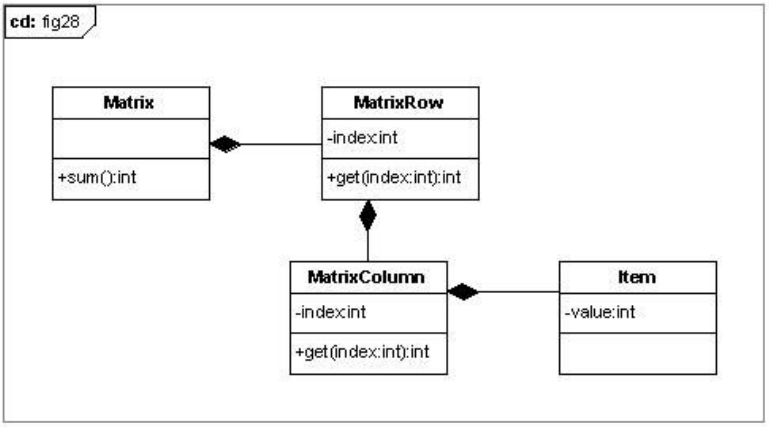
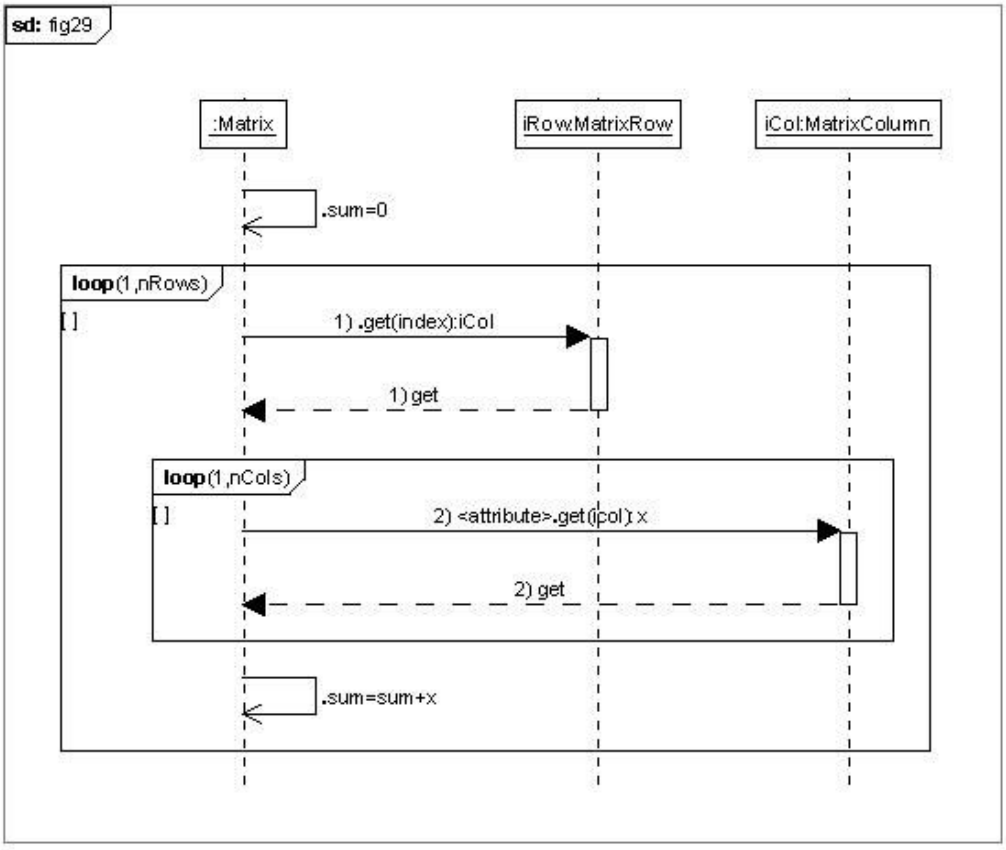


Figure 29



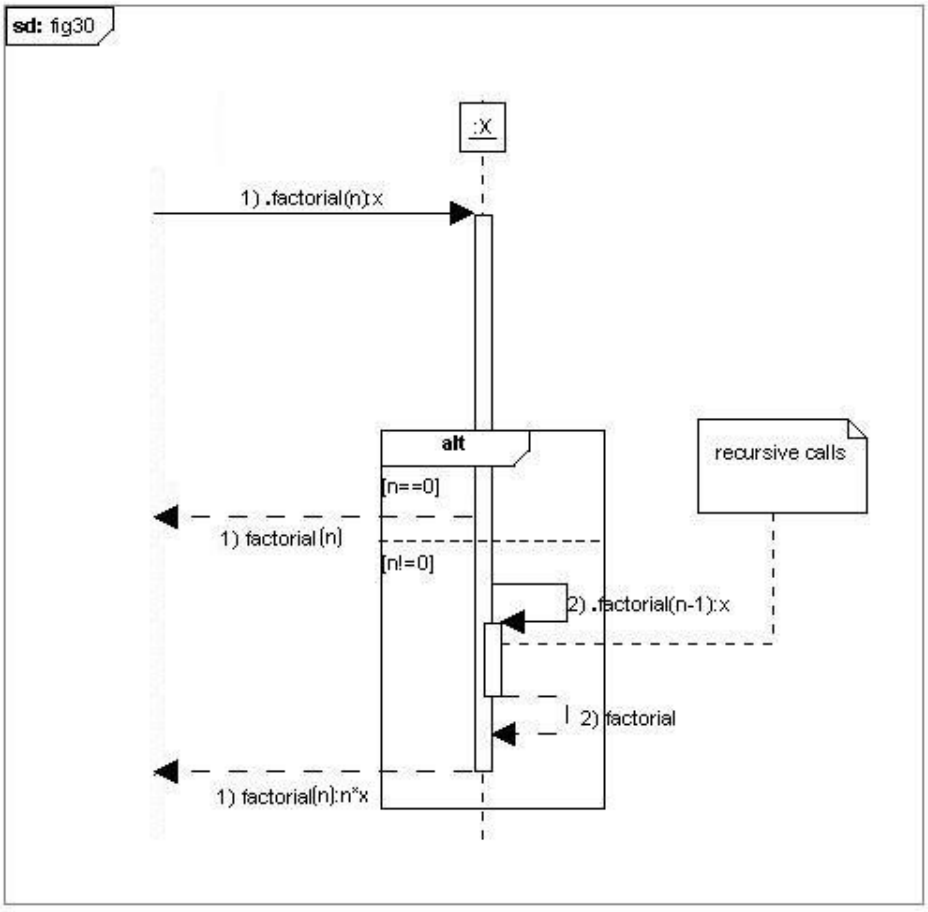
More on the Alt Interaction Operator

The alternatives allow us to select a behaviour function of the logic condition placed before each user. The interaction operator is alt.

We may include an else type logic condition that the associated operator executes when all the other conditions have been proved in the assessment stage to be false.

Figure 30 presents an example of alternative structure for the calculus of the factorial (if …. Else), adapted after [ ].

Figure 30



Remarks: The name of the lifeline, x, is fictional.

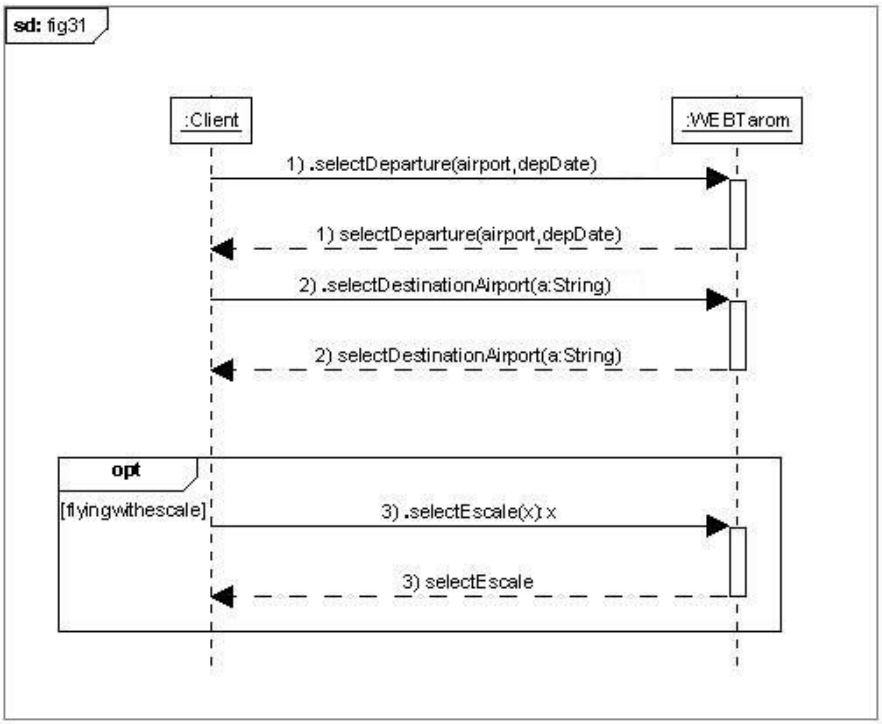
More on the Opt Interaction Operator

Options are interaction backgrounds that are executed only as a result of the assessment of the logic condition, its value is true.

The interaction operator is opt. Conceptually speaking, the options are similar to an alt operator that contains a single user.

Figure 31 presents an example in which the opt operator is used.

Figure 31



The Communication Diagram

The communication diagram (of collaboration in UML 1.x) is an alternative to the sequence diagram. It focuses on a spatial representation of objects. The communication diagrams are not as expressive as the sequence diagrams. When they are used, objects are represented by means of rectangles, and the links between them are represented by continuous lines. Each message possesses a number of sequence: a small arrow that indicates the direction of the message during the respective connection. The communication diagrams do not allow us to represent interaction fragments. Neither do they allow us to represent the order in which the messages are received. The communication diagram is assessed by modelators in the conception stage by its concision and its capacity to imply numerous objects.

Figure 33 illustrates the communication diagram that corresponds to the sequence diagram in figure 32.

Figure 32

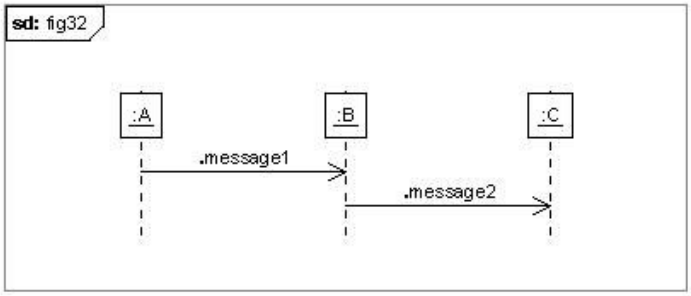


Figure 33

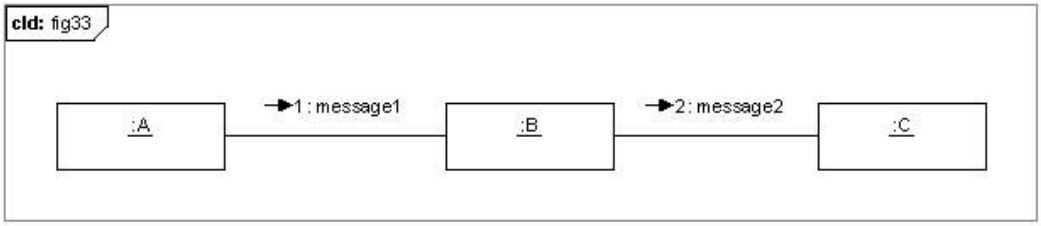
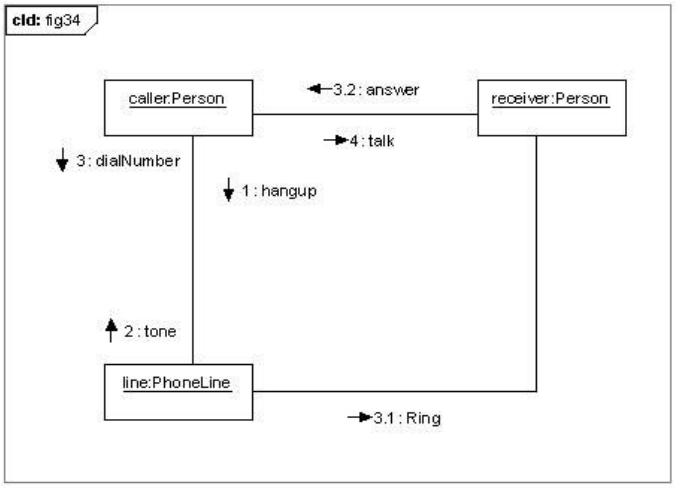


Figure 34 presents the communication diagram, adapted after [ ] for a simplified system of using a phone set that is functional by inserting coins.

Figure 34



Remark: The communication is achieved the moment the receiver answers (message 4).

Figure 35 (a, b, c) illustrates the relation between a class diagram and the sequence diagram and the communication diagram that correspond to it in order to send a message that contains an attachment.

Figure 35a

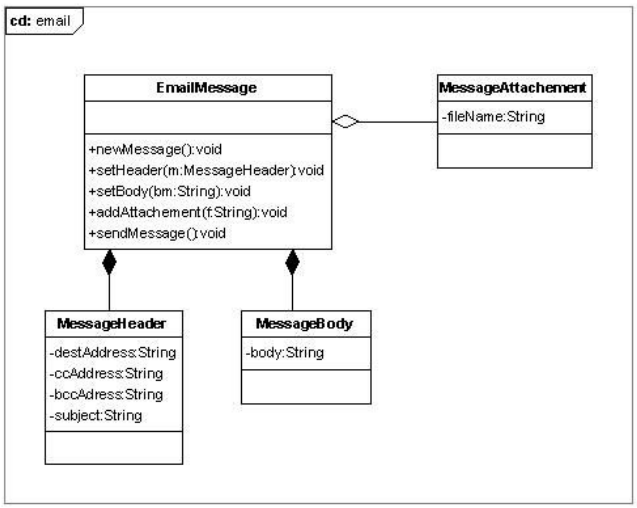


Figure 35b

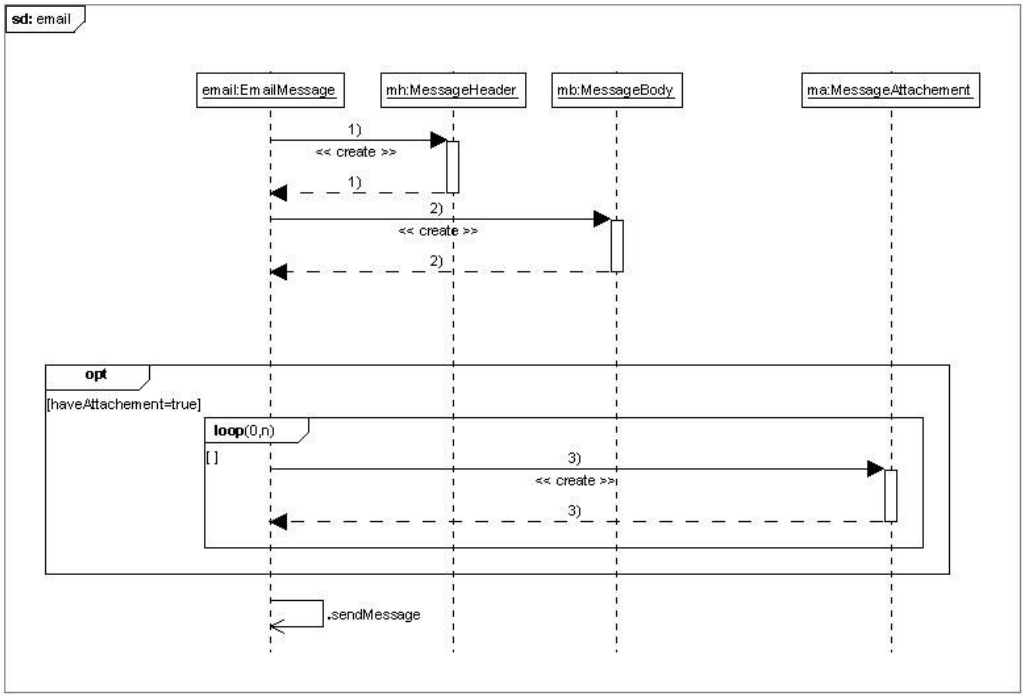
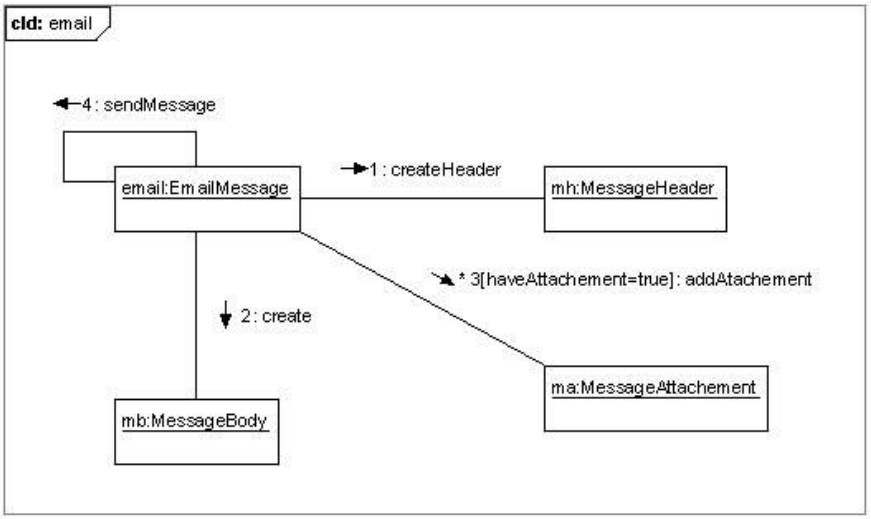


Figure 35c



In a communication diagram, the lifelines are represented by rectangles containing a label whose syntax is represented in figure 36.

Figure 36

<roleName>: <typeName>

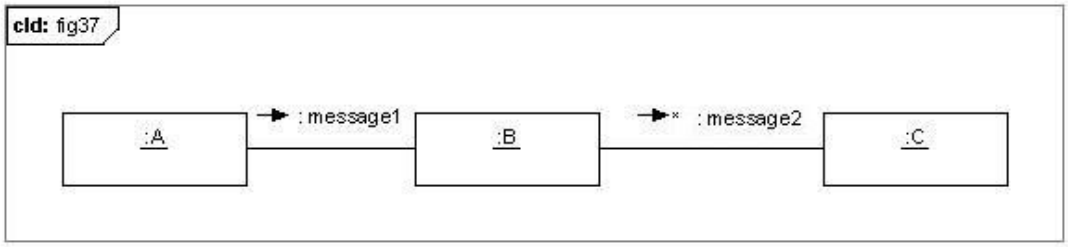
Remarks:

- We need to mention at least two names inside the label (if the role name is omitted, the symbol ":" is compulsory).
- The role (<Rolename>) defines the context in which the interaction is used. If the lifeline is an object, this can contain more roles.
- The communication diagrams are similar to the class diagrams, yet an increased dynamic feature is added to the later ones.

In a communication diagram, the relations between lifelines are called <connectors> (in a class diagram they are called <associations>). A connector is represented in a similar way an association is represented; however, its semantics is larger: the associations appearing in a class diagram represent relations that are independent of the context, while for a system that is in a continuous state of change, it is possible to connect its elements in more ways. For example, an object can be transmitted as a parameter, as a procedure or as a local variation.

Connectors can illustrate the multiplicity of lifelines that participate in an interaction (see figure 37).

Figure 37



Messages

In a communication diagram messages are represented by a continuous line that links the communication partners. Two parallel arrows with the open end indicate the message direction.

The complete syntax of the messages is presented in figure 38.

Figure 38

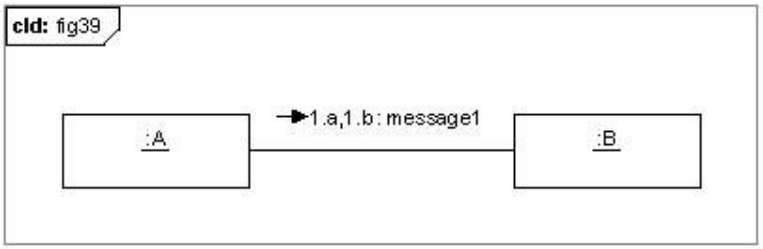
[<sequenceNumber>][<expression>]: <message>

Remarks:

- <message> has the same significance as in the case of sequence diagrams.
- <sequence_number> is an index associated to messages, their counting starts from 1. In order to represent the imbricated message calls, we may add to the signal message number a decimal value that allows us to identify the subsequences (see figure 38).
- <expression> refers to an iteration or a decision.

In a communication diagram concurrent messages can be represented by means of sequence letters (9a, b) (see figure 39).

Figure 39



You draw communication diagrams in the same way as you draw sequence diagrams, the only real difference is that you lay out the notation in a different manner. Messages are added to the associations and show as short arrows pointing in the direction of the message flow. The sequence of messages is shown through a numbering scheme.

As in a communication diagram we cannot represent interaction backgrounds, UML 2 presupposes test mechanisms and loop mechanisms at the level of message sending.

Test mechanisms are achieved by a condition that is specified between square brackets after the message number.

As we have previously mentioned, in a communication diagram, we are not allowed to use interaction backgrounds. However, in a communication diagram we may use loops and decisions, using a peculiar syntax (see figures 40 and 41).

Figure 40

**[<iterationClause>]*

Remark: The iteration clause represents a loop and can be expressed under the form: i:=1..n.

Figure 41

[<selectionClause>]

Remark: The selection clause represents a decision and is a Boolean condition.

Figure 42 presents a generic example of a communication diagram, whose messages are explained in table 2.

Figure 42

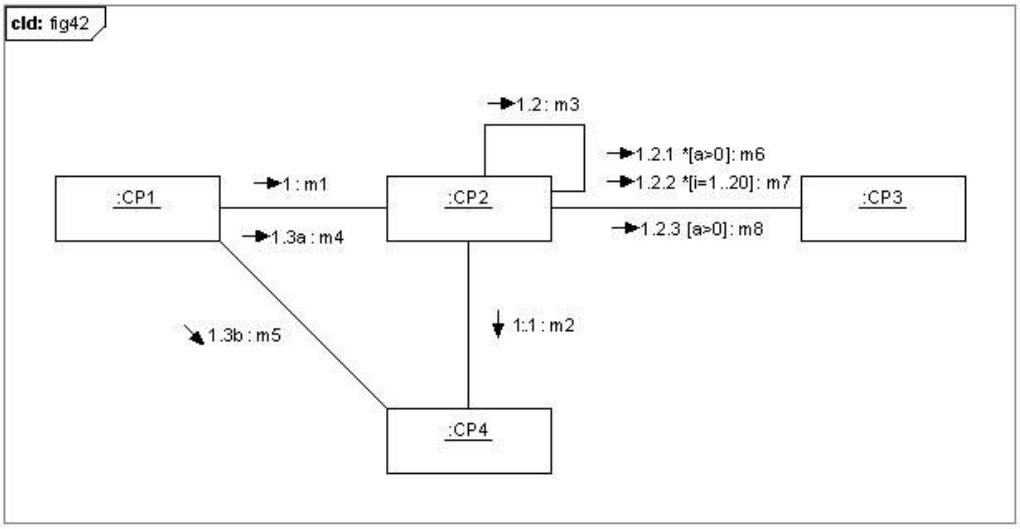


Table 2

Message	*Significance*	*Notes*
1.1: m2	*The first message called inside message 1: m1*	*Cp1, cp2, cp3 are communication partners (lifeline)*
1.3a: m4 and 1.3b: m5	*Messages that can be operated on in parallel.*	*The order is not determined.*
**[a>0]*	*Message m6 is repeated as long as a>0.*	*The minimum number of iterations is zero, and the maximum one is infinite.*
**[i= 1..20]*	*Message m7 was operated on twenty times*	
[a>0]: m8	*Message m8 cannot be operated on but if a>0.*	

The Interaction Overview Diagram

Interaction Overview Diagrams (the diagram that looks at the total number of interaction) are a combination between activity diagrams and sequence diagrams.

The Interaction Overview Diagram is a new type of UML2 diagram and may be included in the category of interaction diagrams, with the sequence, communication and timing diagrams. Consequently, it can be represented by the help of the sd tag.

Interaction Overview Diagrams can be considered activity diagrams in which actions are replaced by interactions.

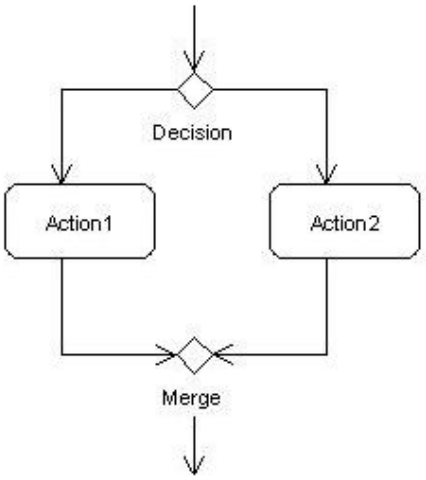
Since an important number of concepts, introduced by sequence diagrams, can be used within Interaction Overview Diagram, we may say that Interaction Overview Diagrams can also be considered sequence diagrams.

The Use of Concepts Implemented in a Sequence Diagram

Within an interaction overview diagram, we may use the following concepts implemented in a sequence diagram:

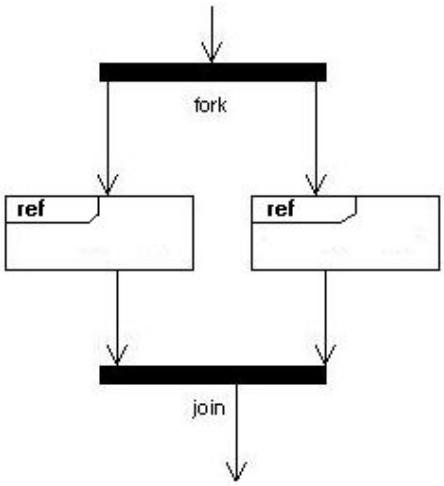
- the representation of combined fragments with the help of a decisional way or a confluence way (see figure 43);

Figure 43



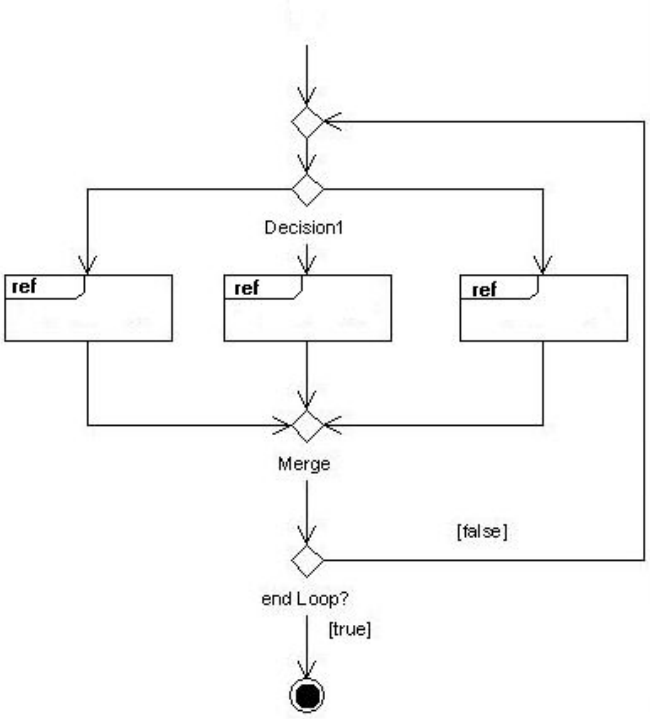
- the representation of parallel interactions with the help of a fork-type nod and a joint-type nod (see figure 44);

Figure 44



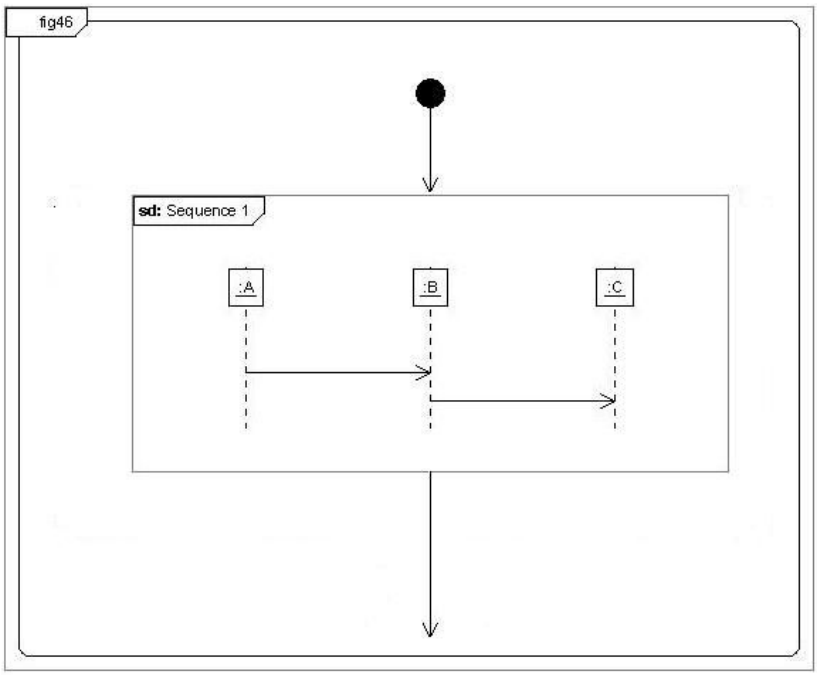
- the representation of loops (see figure 45);

Figure 45



- maintaining the name of lifelines involved in an interaction, with the help of the keyword lifelines (see figure 45);

Figure 46



Timing Diagrams

Notation

Timing diagrams unify state and sequence diagrams in order to describe the evolution of an object state in time and of messages that change this state.

They are used to explore the behaviors of one or more objects throughout a given period of time. It can also show the interaction between timed events and the time and duration constraints that govern them.

Timing diagrams represent another form of interaction diagrams, in which the most important ones are considered the time constraints for one or more objects. Timing diagrams are an innovation brought by UML2.
They are useful to represent temporal restrictions between the change of state for different objects.

Most of the times, timing diagrams are used to model informatics systems in real time conditions.

Unlike the sequence diagrams, timing diagrams are read from left to right. They are not read from the top to the bottom of the page. The timing diagram is represented in two dimensions:

- the vertical dimension, that indicates the lifelines (communication partners) and their state;
- the horizontal dimension which indicates the time axis, that can be gradated.

The timing diagram is a good alternative of class and state diagrams.

Example of a Timing Diagram

Figure 47 presents an example of timing diagram, adapted after [ ], that models parking a car in a parking space with a parking meter.

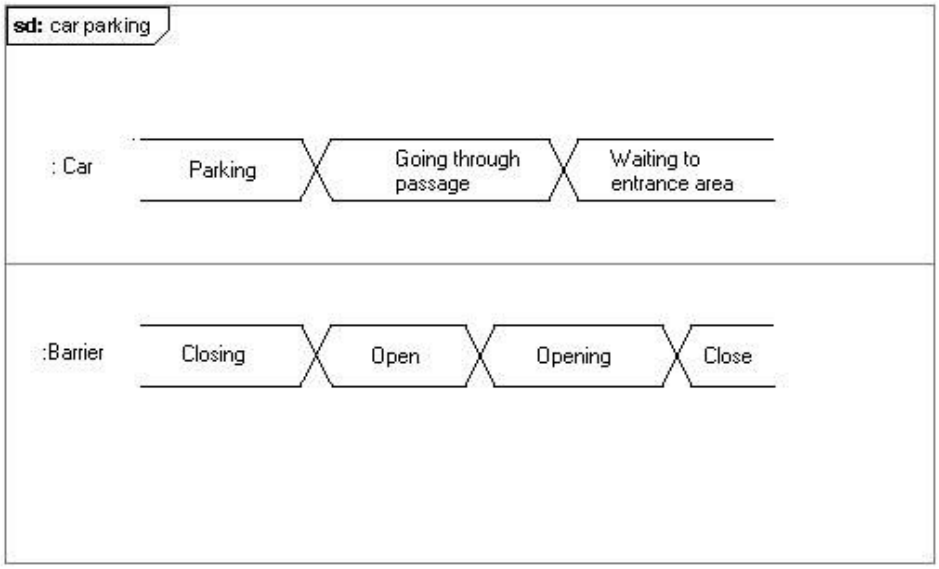
The vertical dimension of the diagram contains the communication partners (the lifelines): the bar, the car and the corresponding states, waiting at the entrance, crossing the parking space, parking the car, opening/ closing the parking bar.

The horizontal dimension of the diagram contains a gradated time axis, in which time intervals are mentioned.

Remarks:

- For each of the communication partners, we have drawn a time line that represent the period in which each lifeline can be found in a certain state. In a horizontal process, the communication partner can be found in the state of the present moment
- (t2-now), and in a vertical (inclined) process, the communication partner changes its state, in this case the duration being significant ({2*d})
- “open”, “free entrance” and “close” are asynchronous messages.
- {t2+2} is a temporal combination.
- “d” is a time interval on the temporal axis.

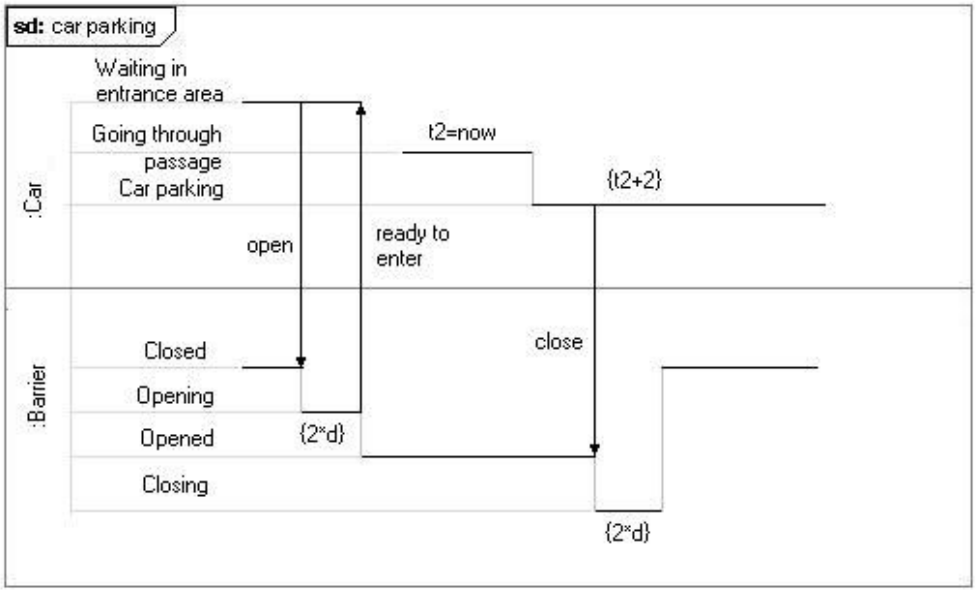
Figure 47



UML 2 offers a simplified notation for temporal lines: the names of states can be mentioned between two horizontal lines which intersect at the moment the states are changed. This notation allows us to represent in an easier way the temporal lines, yet it does not allow us to represent the messages that make the transitions.

Figure 48 illustrates the lifelines: the car, the bar and their different states (see also figure 47).

Figure 48



References

1. Balzert, H. - *UML 2 compact*, Eyrolles, 2006
2. Charroux, B., Osmani, A., Thierry-Mieg, Y. - *UML 2*, Pearson Education France, Collection Syntex, 2005
3. Larman, C. - *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*, Prentice Hall, 1997
4. Pilone, D., Pitman, N., - *UML 2 en concentre*, O,Reilly, Paris, 2006
5. Roques, P. - *UML 2 par la pratique*, 5-e édition, Eyrolles, 2006

Ciclul: UML PENTRU MANAGERI (VII)
Diagrame de interacțiune

Rezumat

Diagramele de interacțiune descriu modul în care obiectele comunică într-un sistem pentru a îndeplini o anumită cerință. Acest articol prezintă conceptele de bază şi conceptele avansate pentru diagramele de interacțiune (UML 2): diagramele de secvență, diagramele de comunicare, diagrame globale de interacțiune şi diagramele de timp.