

# Octopus/UML Notation Summary

Version 1.3

Farfarakis Rallis  
Nokia Research Center

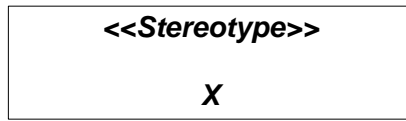
1.	INTRODUCTION .....	1
2.	NOTATION SUMMARY .....	2
2.1	Class Diagram Notation .....	2
2.2	Collaboration Diagram <sup>(A4)</sup> Notation .....	4
2.3	Statechart Diagram Notation .....	6
2.4	Sequence Diagram <sup>(A10)</sup> Notation.....	7
2.5	SDL/GR Process Diagram Notation.....	8
2.6	Outline syntax.....	11
2.7	Special Octopus Features .....	13
3.	BIBLIOGRAPHY.....	14

## 1. INTRODUCTION

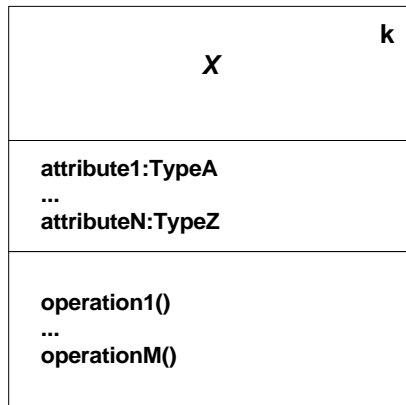
This chapter presents the summary of Octopus/UML notation that also adopts the Unified Modeling Language (UML) [1]. Any superscript, found in the text part of an adornment, indicates a notation enhancement. The interpretation of these superscripts is provided in Chapter Release of Octopus/UML. Six different diagram notations are described: the class diagram, the collaboration diagram, the statechart diagram, the sequence diagram, SDL/GR process diagram and outline syntax. The last section summarises the Special Octopus Features.

## 2. NOTATION SUMMARY

### 2.1 CLASS DIAGRAM NOTATION

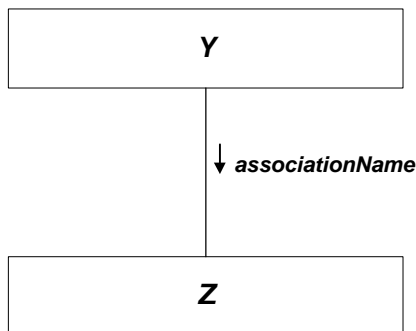


A class labeled with class name *X*. Optionally, the *Stereotype* name is shown.

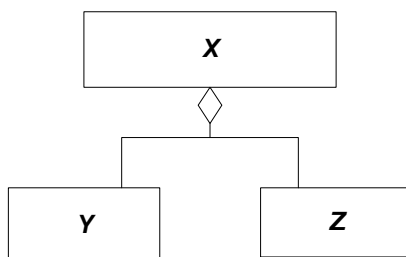


A class with three compartments:

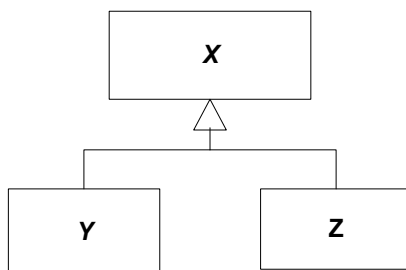
1. The top name compartment declares the class name and optionally the multiplicity indicator *k*.
2. The middle list compartment holds a list of attributes consisting of the name of the attribute and optionally, the type of the attribute. The list compartment is optional.
3. The bottom list compartment contains a list of class-operations consisting of the name of the operations. The list compartment is optional.



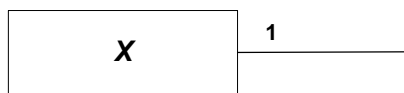
Class *Y* has an association with class *Z*. The association is labeled with a name *associationName* and with an optional arrow, indicating the direction of reading.



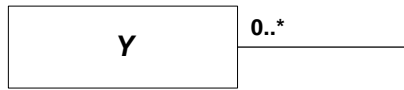
Class *X* aggregates classes *Y* and *Z*.



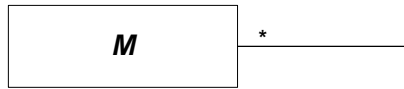
Classes *Y* and *Z* are sub-classes of super-class *X*.



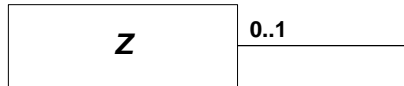
The cardinality of the association is exactly one.



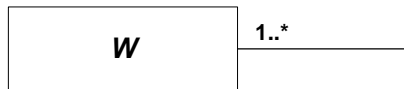
The cardinality of the association is zero or more.



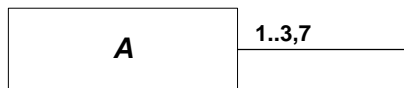
Same as above.



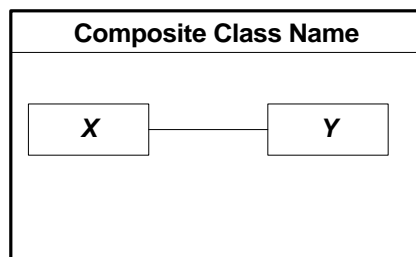
The cardinality of the association is zero or one.



The cardinality of the association is one or more.

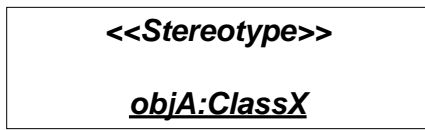


The cardinality of the association is numerically specified.

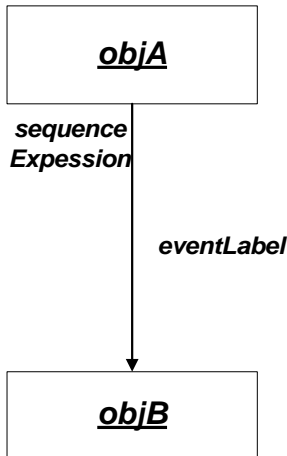


A composite class with other classes as components.

2.2 COLLABORATION DIAGRAM<sup>(A4)</sup> NOTATION



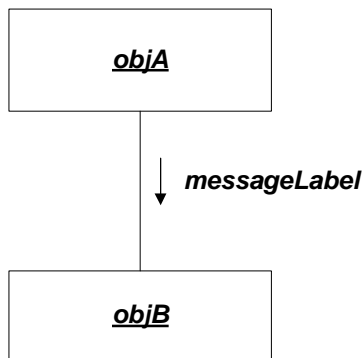
An object with object identifier *objA* and class name *ClassX*. The object identifier, the class name including the colon or the *Stereotype* name are optional.



An unqualified event<sup>(B12)</sup> from object *objA* to object *objB*. The *sequenceExpression* is optional. The syntax of *eventLabel* is:

*EventID: eventName*

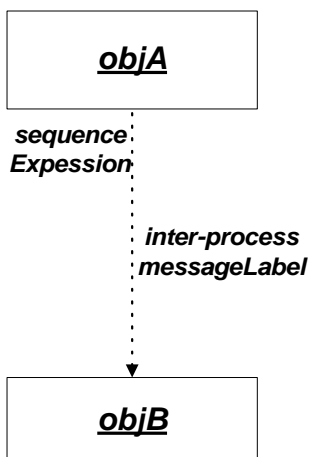
The number of the event ID becomes the first number of the sequence expression of all subsequent message flows. An SOF<sup>(C28)</sup>.



A message flow from object *objA* to object *objB*. If the diagram is qualified<sup>(B13)</sup>, the message flow is implemented by a member function call. The syntax of *messageLabel* is:

*sequenceExpression: messageName*

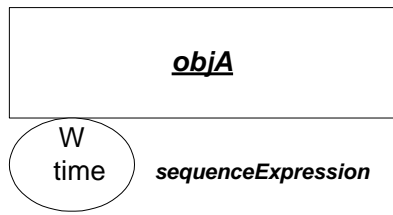
The *sequenceExpression* is a dot-separated list of sequence integers. Each integer represents a level of message flow within the overall interaction.



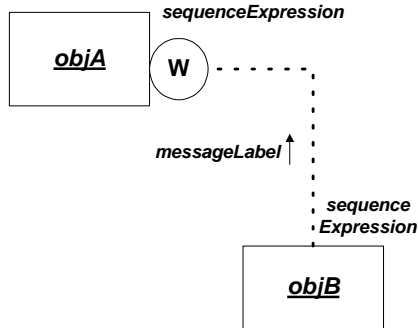
A message flow implemented by an inter-process message with the reception at primary wait point<sup>(B12)</sup>. The *sequenceExpression* is optional. The syntax of *inter-process messageLabel* is:

*inter-process message ID: messageName*

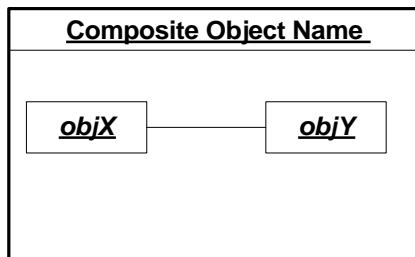
The number of the inter-process message ID becomes the first number in the sequence expression of all the subsequent message flows. An SOF<sup>(C28)</sup>.



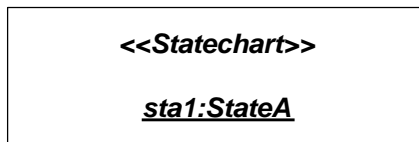
An internal delay at an object *objA*. The indication of time is optional. An SOF<sup>(C28)</sup>.



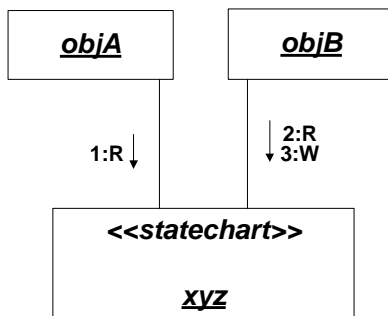
A message flow implemented by an inter-process message with the reception at internal wait point <sup>(B12)</sup> at object *objA*. An SOF<sup>(C28)</sup>.



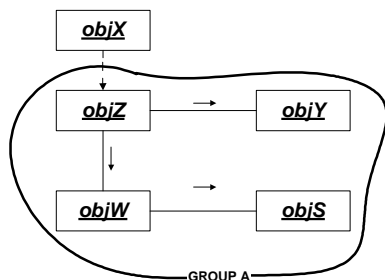
A composite object represents a high-level object made of tightly bound parts.



A statechart object <sup>(B13)</sup> with identifier *sta1* and statechart name *StateA*. Either the identifier or the name of the statechart including the colon, are optional.



The object *objA* reads (R) the current state from statechart *xyz* whereas, the object *objB* reads (R) and then writes (W) a new state in *xyz* <sup>(B13)</sup>.

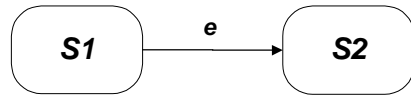


An object group. The freeform line encloses all the objects that belong to *GROUP A*. An SOF<sup>(C28)</sup>.

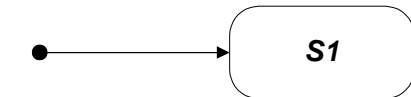
2.3 STATECHART DIAGRAM NOTATION



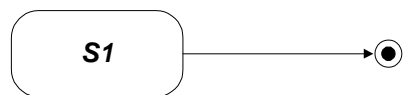
A state labeled with the state name *S1*.



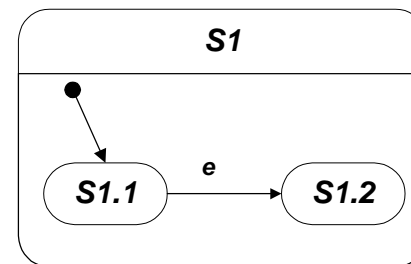
A transition from state *S1* to state *S2* caused by an event *e*.



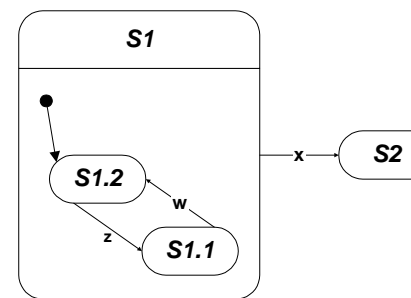
A transition from the initial (pseudo) state to state *S1*.



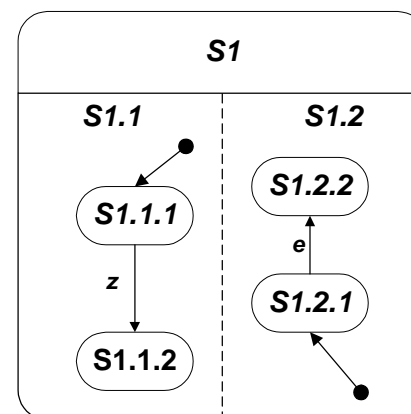
A transition from state *S1* to the final (pseudo) state.



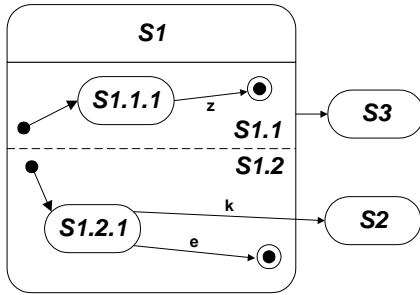
A super-state *S1* nests sub-states *S1.1* and *S1.2*. If the system is in the upper super-state, it is in one and only one of the nested states.



A superstate *S1* is left irrespective of the substate in which it was when the event *x* occurred.

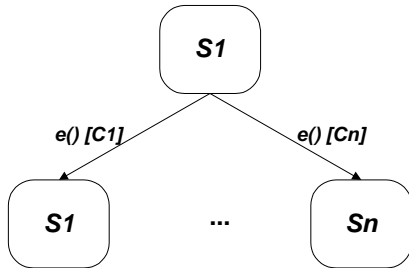


A super-state *S1* has two concurrent regions *S1.1* and *S1.2*. If the system is in *S1* state, it is in a combination of two sub-states: one from region *S1.1* and the other from region *S1.2*.



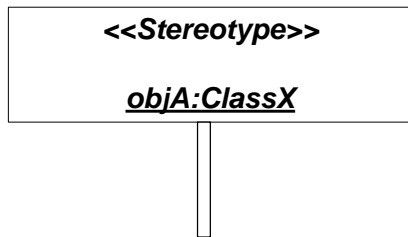
A transition *k* from substate *S1.2.1* to state *S2* outside the enclosing superstate terminates all substatecharts (*S1.1* & *S1.2*) of that superstate.

An unnamed transition from the boundary of superstate *S1* to state *S3* indicates that all concurrent substatecharts (*S1.1* & *S1.2*) must have reached their terminal substates for the transition to occur. Only one unnamed outgoing transition is allowed for each superstate.

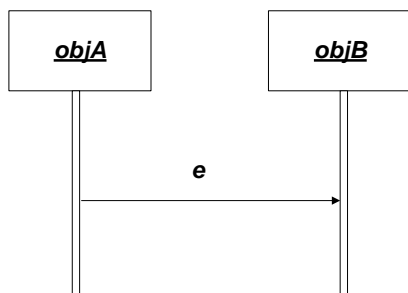


An event *e* causes a transition to one of the states from *S1* to *Sn*, depending on which of the conditions from *C1* to *Cn* are true respectively. Only one condition can be true at a time. One of these conditions is the default.

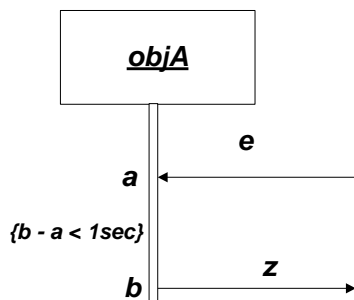
## 2.4 SEQUENCE DIAGRAM<sup>(A10)</sup> NOTATION



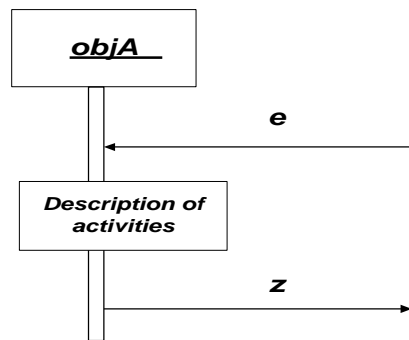
An object with object identifier *objA* and class name *ClassX* with its lifeline. The object identifier, the class name including the colon or the *Stereotype* name are optional.



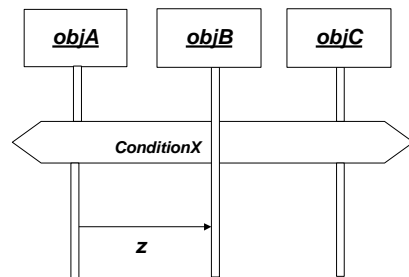
A message *e* is sent by the object *objA* and it is received by the object *objB*.



Labels such as timing marks can be shown either in the margin or near the transitions or activations that they label.



A description of activities can be given upon the reception of *e* message and the sending of *z* message. An SOF<sup>(C28)</sup>.

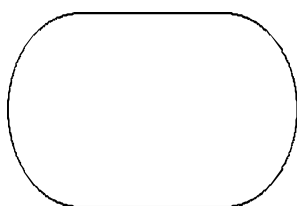


If the condition *ConditionX* is true, then object *objA* sends a message *z* to object *objB*. Both object *objA* and *objC* participate in the condition but not object *objB*. An SOF<sup>(C28)</sup>.



An object activation is shown as a tall thin rectangle. It represents the duration of an action in time and the control relationship between the activation and its callers.

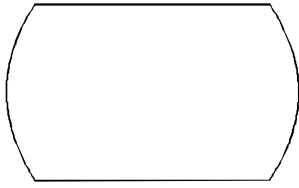
## 2.5 SDL/GR PROCESS DIAGRAM NOTATION



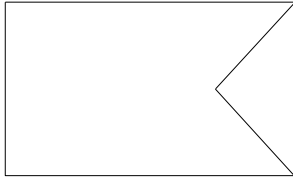
A process start symbol. There is one and only one start symbol per process diagram.



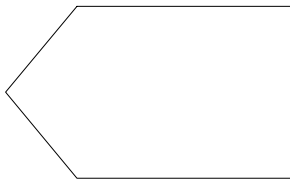
A procedure call symbol. It has one and only one entrance and one and only one exit.



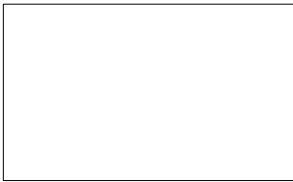
A state symbol. States are designated by a state name.



An input symbol. Input symbols specify the signal name and also to which local variables the signal parameters shall be assigned.



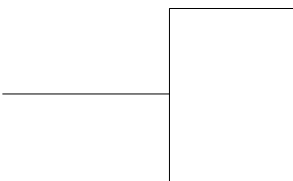
An output symbol. Output symbols specify the signal names and the value of signal parameters.



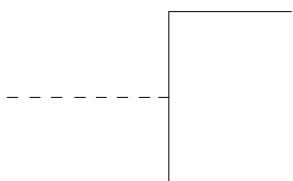
A task symbol. A task is used to set the values of data by assignment.



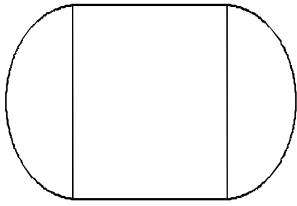
A procedure reference symbol. It contains the name that refers to a procedure definition in a separate diagram.



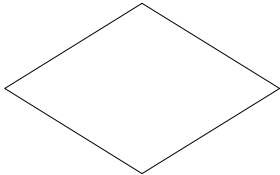
A text extension symbol. It represents an extension of the name within the symbol attached.



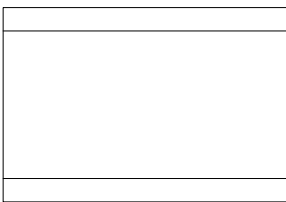
A comment symbol. It shows comments related to the symbol attached.



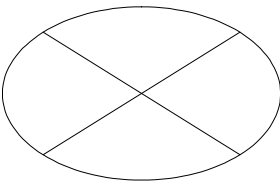
A procedure start symbol.



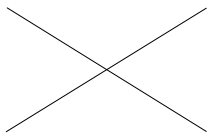
A decision symbol. It is used to choose between alternative courses of action upon a question.



A create request symbol. It causes a process instance to be created and given initial parameter values.



A procedure return symbol. It represents the return from a procedure.



A process stop symbol. It represents the end of a process.



A text symbol. It contains declarations and informal text.

## 2.6 OUTLINE SYNTAX

```

class outline ::=
    class ClassName base spec - opt
        data declaration - opt ; ...
        virtual function declaration - opt ; ...
        function declaration - opt ; ...
    endclass;

process outline ::=
    process function ProcessCodeName
        data declaration - opt ; ...
        execution statement ; ...
    endprocessfunction;

isr outline ::=
    interrupt service routine ISRname
        execution statement ; ...
    endisr;

base spec ::=
    inherits ClassName - list

data declaration ::=
    static - opt type spec - opt identifier array spec - opt /
    statechart StatechartName - opt identifier

type spec ::=
    SimpleTypeName / ClassName / ref ( type spec ) /
    ClassName ( type spec )

array spec ::=
    [ LowerIndex : UpperIndex ]

virtual function declaration ::=
    virtual function functionName ( parameter - opt )

function declaration ::=
    static - opt function functionName ( parameter - opt )
        data declaration - opt
        execution statement ; ...
    endfunction;

parameter ::=
    type spec - opt identifier , ...

execution statement ::=
    VerboseStatement / object spec = expression / return expression /

```

```

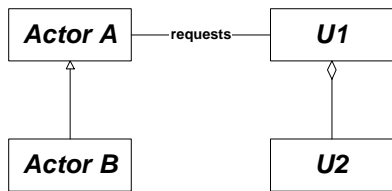
statechart spec = statechart value |
call statement | if statement | switch statement | loop statement
call statement ::=
    functionName ( expression - opt , ... ) |
    object spec . functionName ( expression - opt , ... ) |
    ClassName :: call statement
if statement ::=
    if expression then execution statement ; ... else case - opt endif
else case ::=
    else execution statement ; ...
switch statement ::=
    switch expression
    CaseLabel : execution statement ; ...
    ...
    otherwise case - opt
    endswitch
otherwise case ::=
    otherwise: execution statement ; ...
loop statement ::=
    loop forever execution statement ; ... endloop |
    while expression do execution statement ; ... endwhile |
    do execution statement ; ... while expression enddo
expression ::=
    Value | VerboseExpression | object spec | ClassName :: |
    statechart expression | expression - opt Operand expression
object spec ::=
    identifier | Identifier [ index spec , ... ] |
    functionName ( expression - opt , ... ) |
    object spec . object spec | ClassName :: object spec
index spec ::=
    expression | expression : expression | for each
statechart expression ::=
    statechart spec |
    expression == statechart value | expression != statechart value
statechart spec ::=
    identifier | statechart StatechartName - opt identifier |
    statechart spec . ComponentName

```

*statechart value ::=*  
 StateName / ComponentName . *statechart value*

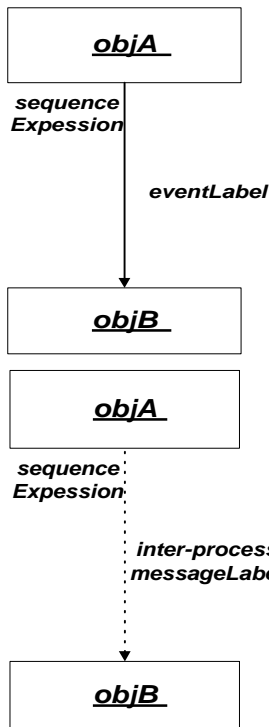
### 2.7 SPECIAL OCTOPUS FEATURES

Using UML notation to describe specific real-time concepts is problematic. The following adornments, termed as Special Octopus Features (SOF)<sup>(C28)</sup>, show a way to model efficiently real-time issues. The SOFs<sup>(C28)</sup> summarised here are extracts from the notation sections above. None of the SOFs<sup>(C28)</sup> is found in UML. Some SOFs<sup>(C28)</sup> are identical with a notation suggested already in the Octopus book [2]. Others, referred as Change or Enhancement ID B12, appear in the list of SOFs<sup>(C28)</sup> because they are enhancements on the original Octopus notation. Although the SOFs<sup>(C28)</sup> are not compliant to UML, a mapping of the SOFs<sup>(C28)</sup> can be adapted in the notation that a tool supports. In future literature, the actual mapping of the SOFs<sup>(C28)</sup> into a number of tools will be presented.



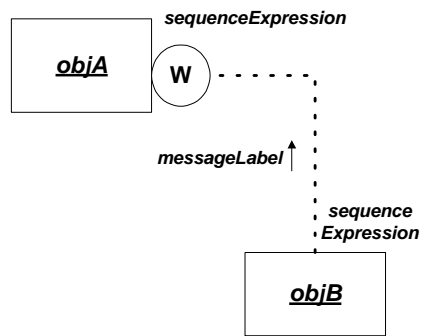
SOF-1.

Standard class diagram is preferred for use case diagrams. UML alternatively allows boxes for actors and use cases.

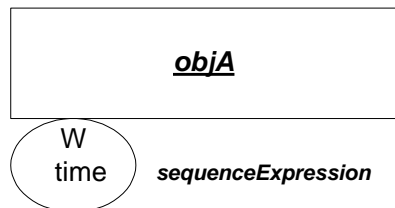


SOF-2. Unqualified event arrow

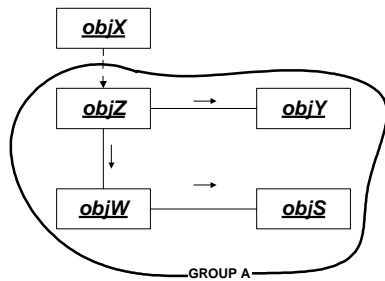
SOF-3. Inter-process message at primary wait point



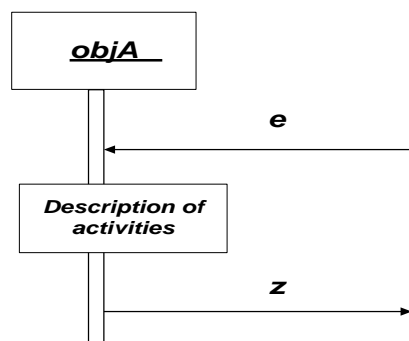
SOF-4. Inter-process message at internal wait point



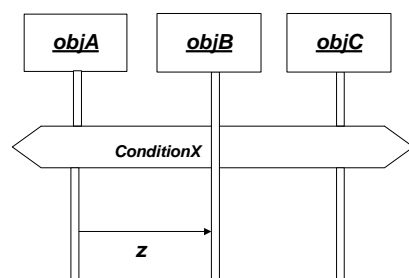
SOF-5. Internal delay



SOF-6. Object group



SOF-7. Activity box



SOF-8. Condition

### 3. BIBLIOGRAPHY

- [1] Unified Modeling Language, Edition 1.1, January 1997.
- [2] M. Awad, J. Kuusela and J. Ziegler. *Object-Oriented Technology for Real-Time Systems*. Prentice Hall, 1996.