

# Octopus/UML Artifact Reference

## Version 1.3

Domiczi Endre, Farfarakis Rallis, Ziegler Jürgen  
Nokia Research Center

|      |  |    |
|------|--|----|
| 1.   | ARTIFACTS AT A GLANCE .....                          | 2  |
| 2.   | DETAILED ARTIFACT DIRECTORY .....                    | 3  |
| 2.1  | Analysis Class Diagram <sup>(A5)</sup> .....         | 3  |
| 2.2  | Class Outline <sup>(B16)</sup> .....                 | 5  |
| 2.3  | Design Class Diagram <sup>(D33)</sup> .....          | 7  |
| 2.4  | Grouped Event Thread .....                           | 8  |
| 2.5  | Input Event Sheet <sup>(A3)</sup> .....              | 9  |
| 2.6  | Inter-Process Message Outline <sup>(A11)</sup> ..... | 10 |
| 2.7  | Inter-Subsystem Scenario <sup>(D25)</sup> .....      | 10 |
| 2.8  | Process Function Outline <sup>(B15)</sup> .....      | 11 |
| 2.9  | Qualified Event Thread .....                         | 12 |
| 2.10 | Responsibility Sheet <sup>(D24)</sup> .....          | 13 |
| 2.11 | Significance Table .....                             | 14 |
| 2.12 | Statechart Diagram .....                             | 14 |
| 2.13 | Subsystem Interface Diagram <sup>(D30)</sup> .....   | 16 |
| 2.14 | Subsystem Operation Sheet <sup>(A2,B11)</sup> .....  | 16 |
| 2.15 | Subsystem Preemption Table <sup>(A7)</sup> .....     | 17 |
| 2.16 | System Architecture Diagram <sup>(A1)</sup> .....    | 18 |
| 2.17 | System Context Diagram .....                         | 19 |
| 2.18 | Unqualified Event Thread .....                       | 20 |
| 2.19 | Use Case Sheet <sup>(B10)</sup> .....                | 21 |

## 1. ARTIFACTS AT A GLANCE

The artifacts are a key concept of Octopus/UML. Each artifact has a purpose, its own semantics, a place within the development sequence, a supporting notation, a mapping of notation-to-semantics and a template. Each artifact is explained in detail in the Section Artifact Directory below. The notation is explained in the Chapter Octopus/UML Notation Summary.

UML defines itself to be a language or notation. Octopus/UML takes this self restriction seriously and strictly distinguishes between the artifacts (or facilities) as work products and the notation used to produce this artifact.

The effort to produce a certain artifact varies. Many of the artifacts are easy to produce and take only few minutes, a few require more sophisticated work to be done and take a few days. For instance, some artifacts can be based on the information already present in existing ones. A copy can be taken and amended accordingly.

For each phase of the development sequence a table is presented (Table 1-1 through 1-4). Each table lists the artifacts of the phase and defines the notation each artifact uses. Artifacts are classified as either basic or advanced. The basic artifacts are always recommended to be produced. Producing the advanced artifacts is necessary only if the size or complexity of the system requires.

The superscripts found in the tables here and elsewhere refer to the change and enhancement IDs defined in the Chapter Release of Octopus/UML.

Table 1-1. Artifacts and Notation for System Requirements Specification Phase

| Artifact                        | Notation      |
|---------------------------------|---------------|
| <b>Basic</b>                    |               |
| Use Case Sheet <sup>(B10)</sup> | Text Table    |
| System Context Diagram          | Class Diagram |

Table 1-2. Artifacts and Notation for System Architecture Phase

| Artifact                                    | Notation         |
|---|------------------|
| <b>Basic</b>                                |                  |
| System Architecture Diagram <sup>(A1)</sup> | Class Diagram    |
| Inter-Subsystem Scenario <sup>(D25)</sup>   | Sequence Diagram |
| Responsibility Sheet <sup>(D24)</sup>       | Text Table       |

Table 1-3. Artifacts and Notation for Subsystem Analysis Phase

| Artifact                                      | Notation           |
|---|--------------------|
| <b>Basic</b>                                  |                    |
| Analysis Class Diagram <sup>(A5)</sup>        | Class Diagram      |
| Subsystem Operation Sheet <sup>(A2,B11)</sup> | Text Table         |
| Input Event Sheet <sup>(A3,B14)</sup>         | Text Table         |
| Statechart Diagram                            | Statechart Diagram |
| <b>Advanced</b>                               |                    |
| Subsystem Interface Diagram <sup>(D30)</sup>  | Class Diagram      |
| Significance Table                            | Text Table         |

Table 1-4. Artifacts and Notation for Subsystem Design Phase

| Artifact                                       | Notation   |
|--|--|
| <b>Basic</b>                                   |  |
| Unqualified Event Thread                       | Collaboration Diagram                            |
| Qualified Event Thread                         | Collaboration Diagram                            |
| Grouped Event Thread                           | Collaboration Diagram                            |
| Class Outline <sup>(B16)</sup>                 | (1) Outline Syntax<br>(2) SDL/GR Process Diagram |
| Process Function Outline <sup>(B15)</sup>      | (1) Outline Syntax<br>(2) SDL/GR Process Diagram |
| <b>Advanced</b>                                |  |
| Design Class Diagram <sup>(D33)</sup>          | Class Diagram                                    |
| Inter-Process Message Outline <sup>(A11)</sup> | Outline Syntax                                   |
| Subsystem Preemption Table <sup>(A7)</sup>     | Text Table                                       |

## 2. DETAILED ARTIFACT DIRECTORY

The detailed artifact directory lists and explains the artifacts in alphabetical order. The phase in which the artifact belongs, and the notation used to express the artifact is given.

In Octopus/UML, the same notation may be used for various artifacts. Therefore, an artifact can make use of the notation in a way that is most suitable for its purpose and context. This mapping of notation and artifact is stated in the directory.

Moreover, a template is given for each artifact. The templates of graphical artifacts are generic examples. The templates of text tables are in their final form and layout, but empty. So, a short explanation is given in each type of field.

Finally, an attachment<sup>(C27)</sup> may accompany the artifact. If an artifact has an attachment, the description, notation, mapping and template of the attachment are also described.

### 2.1 ANALYSIS CLASS DIAGRAM<sup>(A5)</sup>

The purpose of the Analysis Class Diagram is to express the structural part of the common understanding the subsystem development team has achieved. It shows, as classes, the concepts

the subsystem deals with, the associations between those concepts, and optionally, the relevant attributes of the concepts. This view should remain valid even though in the design phase the developer may produce a Design Class Diagram.

The Analysis Class Diagram also shows the boundary between the subsystem and its environment. The classes outside the subsystem boundary are called the environment classes and they express concepts from other subsystems or whole subsystems. Of course, the analysis class diagram of such a subsystem must show a consistent mirror image.

The Analysis Class Diagram can be a single one-page diagram or a set of linked diagrams. For example, it might be worthwhile to produce class diagrams at different abstraction levels, and for different points of view. Further, if a class diagram exceeds normal page size, it should be broken into a set of hierarchically linked ones.

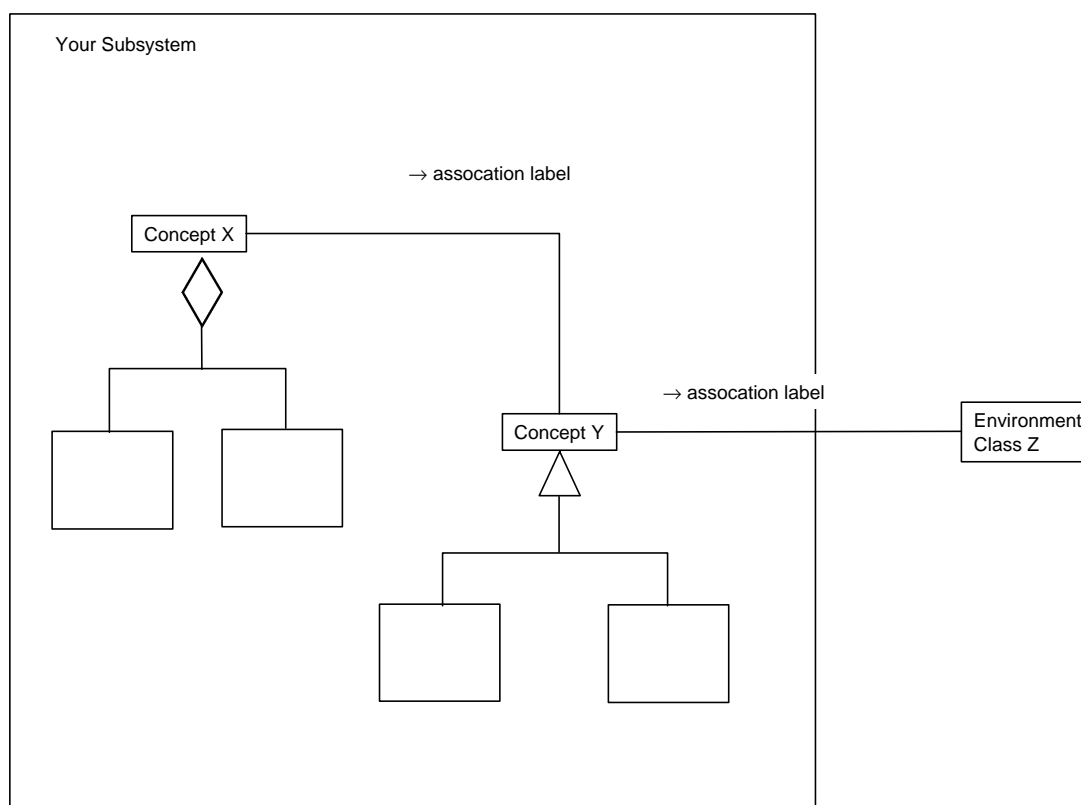
The Analysis Class Diagram may also use an attachment<sup>(C27)</sup> termed as Analysis Class Description Table.

**Phase:** subsystem analysis

**Notation:** class diagram notation

**Mapping:** problem domain concepts as classes; subsystem as composite class; relationships between concepts as associations between classes

**Template:**



**Attachment:** Analysis Class Description Table<sup>(A9)</sup>

The purpose of the Analysis Class Description Table is to provide a textual description of all concepts shown inside the subsystem in the Analysis Class Diagram. Also the associations and the optionally shown attributes must be described here. This table complements the Analysis Class Diagram. It should provide all the detailed information needed when, during the study of the class diagram, an item requires further clarification. If an association crosses

the boundary, it must be described here and in the complementary artifact of the target subsystem. The descriptions must be consistent.

The Analysis Class Description Table also provides the information of the environment classes, that exist outside the subsystem boundary and they express concepts from other subsystems or whole subsystems. Any association crossing the subsystem boundary of the analysis class diagram defines a static relation with an environment class of another subsystem. The benefit of this attachment is effective and consistent communication between subsystems.

**Notation:** text table notation

**Mapping:** two columns; floating number of rows; column headers in first row; one row for each class; two parts, one for subsystem classes, one for environment classes

**Template:**

| Class               | Description  |
|---------------------|--|
| Subsystem Classes   |  |
| Concept X           | Concise and understandable description of Concept X  |
| attribute 1         | Description of attribute 1   |
| attribute 2         | ...  |
| association 1       | Description of the association of Concept X with Concept Y; always state which is client and which server; each association must appear at least once in a class |
| association 2       |  |
| Environment Classes |  |
| Concept X           | Extract from description of other subsystems that class belongs to. Reference to the full description  |

## 2.2 CLASS OUTLINE<sup>(B16)</sup>

The purpose of the Class Outline is to become the baseline for the implementation of classes in the selected programming language. Optionally, the Class Outline can be termed either initial or final Class Outline depending on what stage the development is when this artifact is used.

The Class Outline declares the structure of the class with all member functions. The artifact captures the details not expressed in event threads. If you select an inter-process message for the implementation of a message flow, the corresponding statement should outline preparation and sending of the inter-process message with all necessary data. If you decided to receive an inter-process message at an internal wait point, the corresponding statement should be correctly placed. If access of shared objects is critical, the class outline must describe how synchronization is made. The member function bodies must outline the final flow of execution but avoid details of computation and flow control.

The standard notation to represent Class Outlines is outline syntax notation.

**Phase:** subsystem design

**Notation Alternative 1:** outline syntax (standard)

**Mapping:** class interface by syntactical keywords and user defined names; member function implementation as body of the member function declaration; member function

implementation statement as natural language sentence and/or syntactical keyword statement

**Template:**

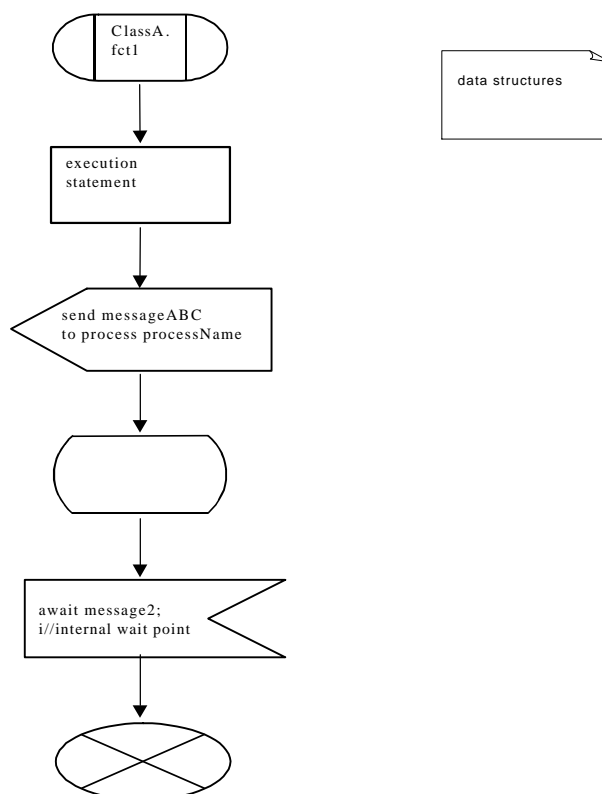
```

class ClassName inherits ClassName-list
  data declaration;
  virtual function functionName(parameter);
  function functionName(parameter)
    data declaration;
    execution statement;
    objXYZ.doThis(parameter);
    //objABC.doThat(parameter) done by inter-process message
    messageABC.parameter = values;
    send messageABC to process processName;
    execution statement;
    await message2; //internal wait point
  endfunction;
endclass;
    
```

**Notation Alternative 2: SDL/GR Process Diagram**

**Mapping:** Class member functions as procedure; flow of execution as standard flowchart; inter-process message sending as output, note for data structures; internal wait point as message receive

**Template:**



### 2.3 DESIGN CLASS DIAGRAM<sup>(D33)</sup>

The purpose of the Design Class Diagram is to capture structural refinements that need to be added to the structural model produced during the analysis phase. These refinements should cover only issues of how to build what has been specified in the analysis phase.

Any class shown in the Analysis Class Diagram either re-appears in the Design Class Diagram or is further decomposed. If a class simply re-appears one to one, the problem domain concept, represented in the Analysis Class Diagram, is transformed into a design class. A design class is the key element for designing the subsystem. The design phase describes how the instances of the design classes, or the objects for short, collaborate. If a class of the Analysis Class Diagram is decomposed, the problem domain concept is transformed into a set of design classes. The mapping must be stated in the Design Class Description Table.

In addition, the Design Class Diagram can show design classes representing pure design issues. In all other aspects, the Design Class Diagram is treated similar to the Analysis Class Diagram.

If no Design Class Diagram is produced, all problem domain concepts are transformed, one to one, into design classes. The subsystem design is based only on those design classes.

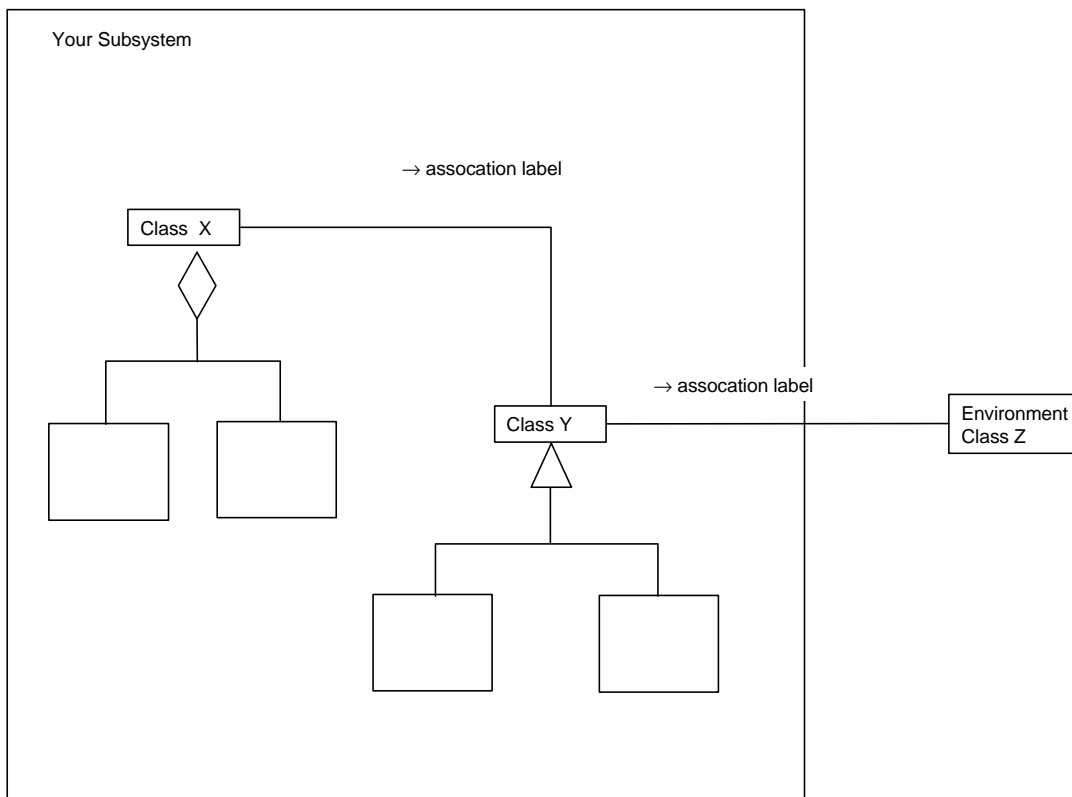
The Design Class Diagram may use the attachment<sup>(C27)</sup> Design Class Description Table.

**Phase:** subsystem design

**Notation:** class diagram notation

**Mapping:** design classes as classes; subsystem as composite class; relationships between design classes as associations between classes

**Template:**



**Attachment:** Design Class Description Table<sup>(D26)</sup>

The purpose of the Design Class Description Table is to provide a textual description of all design classes shown inside the subsystem in the Design Class Diagram. In addition, the

associations, the data members and member functions must be described here. However, if a problem domain concept has been translated one-to-one into a design class, a description can be omitted.

This table compliments the Design Class Diagram. It should provide or at least refer to, all the detailed information needed, when during the study of the class diagram, an item requires further clarification. If the design class is the result of a decomposition of a problem domain concept, a mapping to that concept of the Analysis Class Description Table must be stated.

If an association crosses the boundary, it must be described here and in the complementary artifact of the target subsystem. The descriptions must be consistent.

**Notation:** text table notation

**Mapping:** two columns; floating number of rows; column headers in first row; one row for each class including attributes and associations

**Template:**

| Class               | Description  |
|---------------------|--|
| Subsystem Classes   |  |
| Design Class X      | Concise but clearly understandable description of Design Class X   |
| data member 1       | Description of data member 1   |
| data member 2       | ...  |
| member function1    | Description of member function 1   |
| member function2    | ...  |
| association 1       | Description of the association of Design Class X with Design Class Y; always state which is client and which server; each association must appear at least once in a class |
| association 2       |  |
| Environment Classes |  |
| Design Class X      | Extract from description of other subsystems that class belongs to. Reference to the full description  |

## 2.4 GROUPED EVENT THREAD

The Grouped Event Thread is based on the corresponding Qualified Event Thread. Its purpose is to define the object groups. An object group starts with one or more inter-process messages and covers all subsequent nested member function calls of each inter-process message.

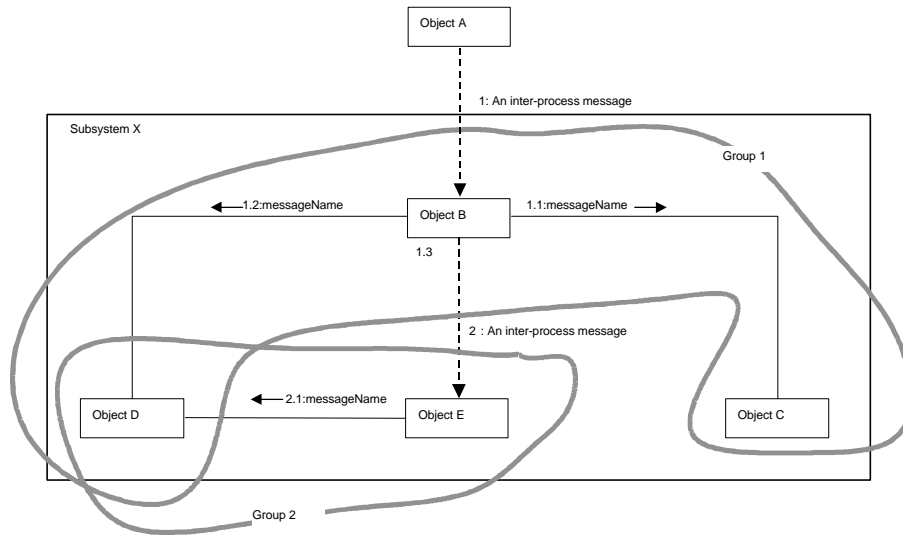
If desirable, any number of Grouped Event Threads may be overlaid in a single figure, and the result is nonetheless readable. In particular, if two or more object groups cover the same object, the grouped event threads should be overlaid in one figure. This overlaying of the object groups reveals the shared object graphically which is then entered into the shared object table.

For each object group exactly one process will be built. This process is outlined with the process function outline.

**Phase:** subsystem design

**Notation:** collaboration diagram with Octopus enhancements

**Mapping:** same as Qualified Event Thread; object group as defined by Octopus

**Template:****2.5 INPUT EVENT SHEET<sup>(A3)</sup>**

The purpose of the Input Event Sheet is to specify all the essential information concerning an input event, for the subsystem under consideration.

An input event announces that something has happened in the environment of the subsystem, that is in another subsystem, and requests the subsystem to react. The input events are identified in the Event List and grouped in the Event Diagram.

Input Event Sheets are produced for the events listed in the Event List and for the superevents defined in the Event Diagram. If the Input Event Sheets for superevents cover adequately the information of the individual events, you need not produce event sheets. If the events of a group have common properties, record this common part only in the event sheet of the group's superevent. This information should not be repeated this in the sheets of the group members.

An event is a primary event<sup>(C24)</sup> if it is originating from the environment wrapper<sup>(C22)</sup>. From the point of view of an application domain subsystem receiving such a primary event, its origin relates to an abstract environment class. This class in turn represents an actor in the environment of the system, as shown in the System Context Diagram.

An event is a secondary event<sup>(C24)</sup> if it originates from an application domain subsystem. A secondary event is a request for service, the originating subsystem being the client, the receiving subsystem the server.

**Phase:** subsystem analysis

**Notation:** text table notation

**Mapping:** two columns; fixed number of rows; row headers in left column

**Template:**

|                      |  |
|----------------------|--|
| <b>Event</b>         | ID + event name  |
| <b>Response</b>      | Description of the desired response from the subsystem, may be execution of a subsystem operation or a set of subsystem operations or part of a subsystem operation or production of an output event |
| <b>Source</b>        | Originator of the event, i.e. the environment class; if this event is a primary event, refer to actor too  |
| <b>Contents</b>      | Data in arbitrary format that may be delivered to the subsystem in connection with the event   |
| <b>Response Time</b> | Maximum and/or minimum time limits of the response   |
| <b>Rate</b>          | Rate of occurrence, for example, at startup, periodic every 10 minutes, timed at 8:00 AM and 2:00 PM, occasional, exceptional, etc   |
| <b>Related to</b>    | Classes, objects, operations or statecharts to which the event may be related  |

## 2.6 INTER-PROCESS MESSAGE OUTLINE<sup>(A11)</sup>

The purpose of the Inter-Process Message Outline is to describe the structure of the inter-process message. Any originator object that wishes to send an inter-process message based on the facilities of the operating system, uses this description to correctly define how to prepare and send the inter-process message in its Class Outline.

If the receiver object of an inter-process message provides a specific software package, that serves as a more convenient interface for preparing and sending the inter-process message, the Inter-Process Message Outline becomes the baseline for the implementation of that interface software. This interface software should be defined using Class Outlines in the usual manner. These Class Outlines may include the Inter-Process Message Outlines. Any originator object that wishes to send an inter-process message via this interface software, uses this description to correctly define how to call the interface software in its Class Outlines.

**Phase:** subsystem design

**Notation:** outline syntax

**Mapping:** message data structure as class with data attributes;

**Template:**

```
class ClassName inherits ClassName-list
  data declaration;
endclass;
```

## 2.7 INTER-SUBSYSTEM SCENARIO<sup>(D25)</sup>

The purpose of the Inter-Subsystem Scenario is to demonstrate a single, explicit trace of interactions between the subsystems and the actors. This trace shows one of the many ways of using the system. It relates to a certain use case or a set of use cases.

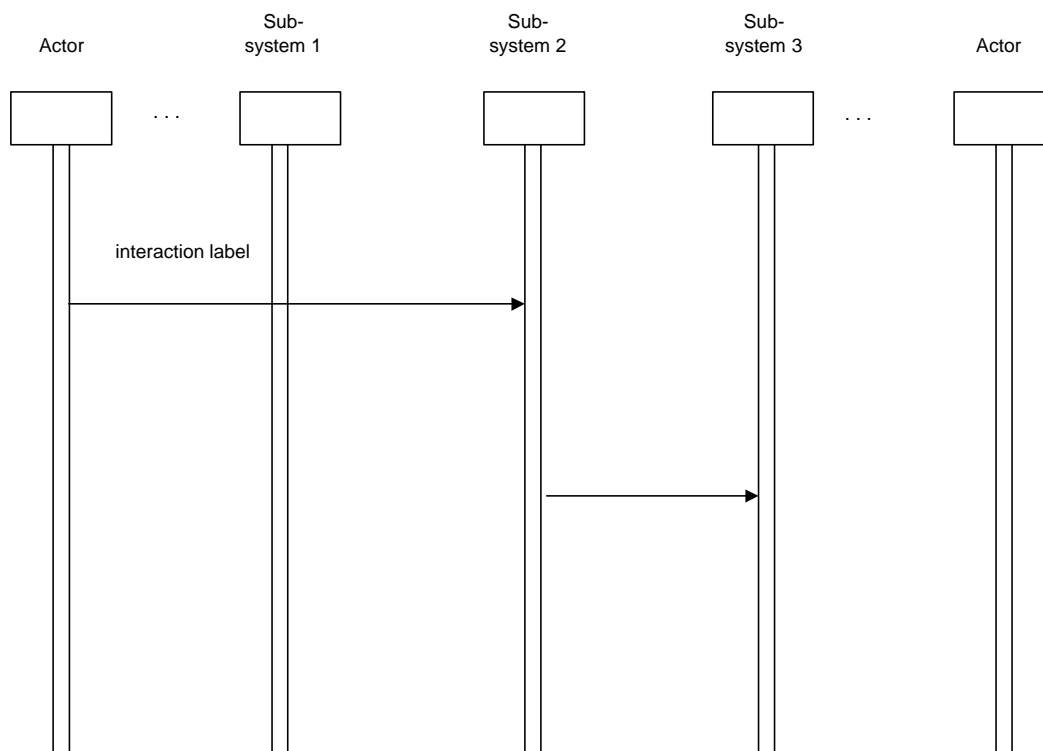
The Inter-Subsystem Scenario may be an explosion of a System Scenario.

**Phase:** system architecture

**Notation:** sequence diagram notation

**Mapping:** subsystems as objects; actors as objects; interactions between actors and the subsystem as messages; all objects shown with concurrent activation along the whole lifeline

**Template:**



## 2.8 PROCESS FUNCTION OUTLINE<sup>(B15)</sup>

The purpose of the Process Function Outline is to become the baseline for the implementation of an object group process, in terms of the real-time operating system the software is based on.

A process function outline applies the recommended standard pattern of object group processes, or a customized version of it, consisting of an infinite loop, a primary wait point and a switch to the original member function call depending on the received message.

The process function outline may also outline an interrupt service routine, since interrupt service routines can be treated as the highest priority processes. If an interrupt service routine is the corresponding process of an object group, the interrupt service routine drops the infinite loop and the wait point of the standard pattern.

**Phase:** subsystem design

**Notation Alternative 1:** outline syntax (standard)

**Mapping:** process implementation as body of function; process implementation statement as natural language sentence and/or syntactical keyword statement

**Template:**

```

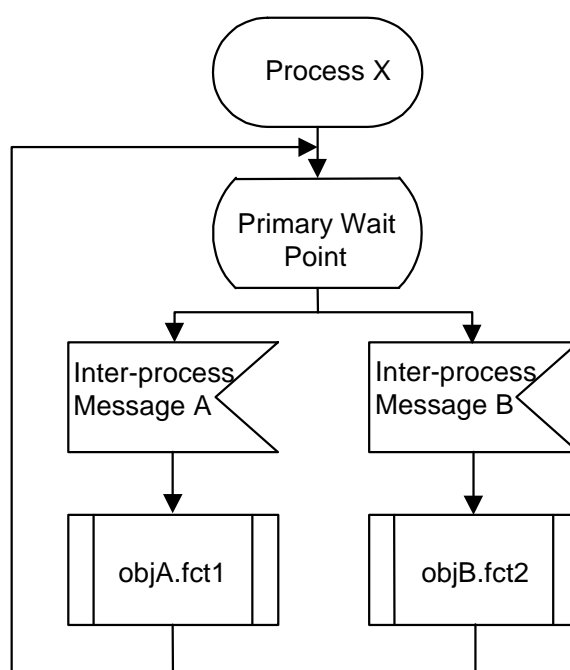
process function ProcessCodeName //corresponds to group X
  loop forever
    await inter-process messages;
    switch kind of message
      messageABC:
        objABC.doThat(message.parameter); //forward to member fct.
      endcase;
    endswitch;
  endloop;
endprocfuction;
interrupt service routine ISRname //corresponds to group Y
  switch kind of interrupt
    intUVW: objUVW.isrEntry();
  endswitch;
endisr;

```

### Notation Alternative 2: SDL/GR process diagram notation

**Mapping:** object group process as process; primary wait point as state with attached inputs; member function calls as procedure call;

**Template:**



## 2.9 QUALIFIED EVENT THREAD

The Qualified Event Thread is based on the corresponding Unqualified Event Thread. Its purpose is to capture our decision about the implementation mechanism of each message flow. Basically, synchronous or asynchronous mechanisms are the only choices. Selecting a synchronous mechanism means that the message flow will be implemented by a conventional, nested member function call. Selecting an asynchronous mechanism defines an inter-process message, which is

entered into the inter-process message list. The inter-process message will be implemented by using the inter-process communication services of the real-time operating system.

The Qualified Event Thread takes processing time into account, including the time of the respective implementation mechanism of the message flow. The processing time is one factor, besides others, affecting the decision about synchronous versus asynchronous.

If an asynchronous mechanism has been selected, and the original message flow included return values, an additional inter-process message for transferring this return value must be defined and shown in the Qualified Event Thread.

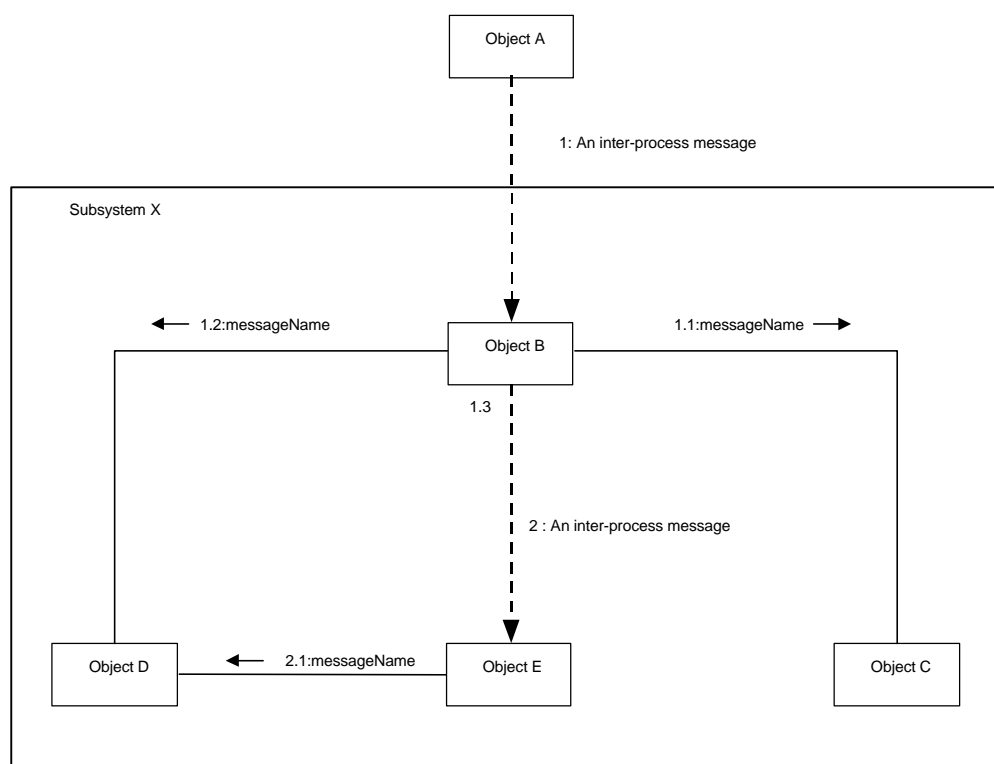
Reception of an inter-process message can be done either in a primary wait point or an internal wait point. The Qualified Event Thread expresses this decision too.

**Phase:** subsystem design

**Notation:** collaboration diagram with Octopus enhancements

**Mapping:** same as unqualified event thread, except that no change of message flow indicates member function call; inter-process message with reception either at primary wait point or internal wait point as defined by Octopus

**Template:**



## 2.10 RESPONSIBILITY SHEET<sup>(D24)</sup>

The purpose of the Responsibility Sheet is to list and to describe all responsibilities of a certain subsystem. It expands the corresponding column of the Responsibility Matrix.

**Phase:** system architecture

**Notation:** text table

**Mapping:** three columns; as many rows as responsibilities of subsystem;

**Template:**

| Subsystem X       |   |   |
|-------------------|---|---|
| Responsibility ID | Description   | From use case                                   |
| ID                | Statement describing this responsibility in terms of the demanding use case | Refer to use case demanding this responsibility |
| ...               | ...   | ...   |

**2.11 SIGNIFICANCE TABLE**

The purpose of the Significance Table is to present how important an event is for all elementary states of the Statechart Diagrams. The significance of events influences the event threads and process priorities in the subsystem design phase.

The Significance Table is used in order to further specify and comprehend the dynamic behaviour of the subsystem under consideration.

**Phase:** subsystem analysis

**Notation:** text table notation

**Mapping:** floating number of columns; floating number of rows

**Template:**

|                   | Event             |                   |                   |       |
|-------------------|-------------------|-------------------|-------------------|-------|
| State             | <ID + event name> | <ID + event name> | <ID + event name> | <...> |
| <ID + state name> |                   |                   |                   |       |
| <...>             |                   |                   |                   |       |

**2.12 STATECHART DIAGRAM**

The purpose of the Statechart Diagram is to describe a set of states and the significant transitions between them. The state transitions occur due to the appearance of certain stimuli. A state expresses satisfaction of a condition, execution of an ongoing activity or the waiting of an event. Typical transition stimuli are input events, but also detection of an internal condition may trigger a state transition.

A statechart diagram is worth building whenever an important, sufficiently complex, state-dependent behaviour is encountered, since it clarifies our understanding. A statechart diagram is neither forced to describe the entire system nor is restricted to a single class. The information presented in the statecharts is highly related to other artifacts such as Subsystem Operation Sheets and Input Event Sheets. This inter-dependency feature should be maintained in the best possible way.

If a state transition requests execution of an action, or a state expresses execution of an ongoing activity, an accompanying Actions Table should list them.

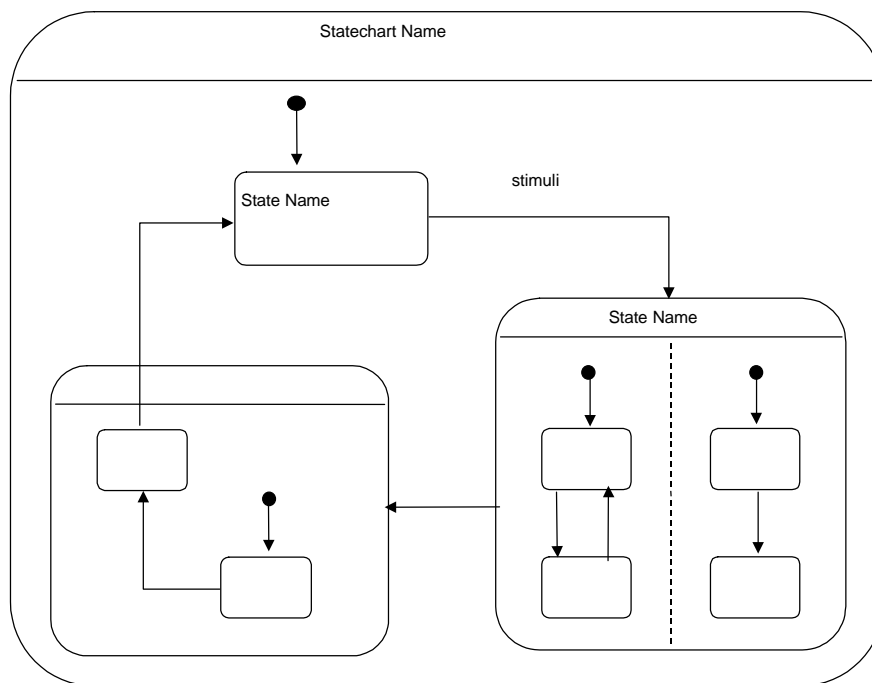
The Statechart Diagram may utilise the attachment<sup>(C27)</sup> Actions Table.

**Phase:** subsystem analysis

**Notation:** statechart diagram notation

**Mapping:** as defined by notation

**Template:**



**Attachment:** Actions Table

The Actions Table is an optional attachment to a Statechart Diagram. Its purpose is to list and describe the actions executed when a state transition is performed, or the activities are going on in a specific state. Putting this information into a separate Actions Table keeps the Statechart Diagram focused on the states and their transitions.

If an action is executed whenever the state is left, regardless of which the target state will be, it can be listed under the <exit> field. If an action is executed whenever the state is entered, regardless of the originating state, it can be listed under the <enter> field. If an activity is in operation in a certain state, it is listed under the <ongoing> field.

**Notation:** text table notation

**Mapping:** three columns; floating number of rows; column headers in top row

**Template:**

| State           | Transition Stimuli                                     | Action/Activity  |
|-----------------|--|--|
| ID + state name | stimuli 1<br><br>...<br><enter><br><exit><br><ongoing> | actions executed during transition triggered by stimuli 1 at this state, i.e. an event<br><br>...<br>actions executed when entering this state<br>actions executed when exiting this state<br>activities ongoing in this state |
| ...             | ...  | ...  |

**2.13 SUBSYSTEM INTERFACE DIAGRAM<sup>(D30)</sup>**

The purpose of the Subsystem Interface Diagram is to ensure that the development team of the subsystem under consideration, has gathered all the necessary knowledge concerning the static relationships between this subsystem and the other subsystems.

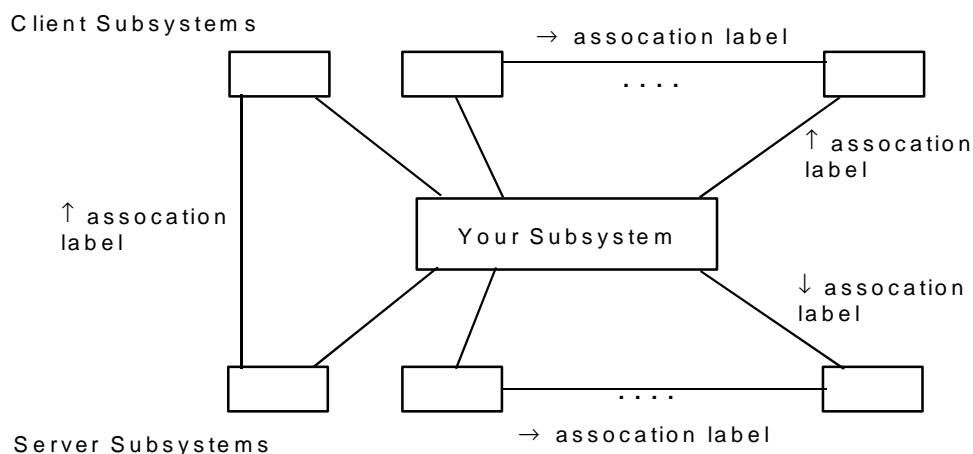
System architecture defines all subsystems and the System Architecture Diagram describes the relationships between them. The Subsystem Interface Diagram is basically an extract from the System Architecture Diagram focusing on one subsystem.

**Phase:** subsystem analysis

**Notation:** class diagram notation

**Mapping:** subsystems as classes; relationships between subsystems as associations between classes

**Template:**



**2.14 SUBSYSTEM OPERATION SHEET<sup>(A2,B11)</sup>**

The purpose of the Subsystem Operation Sheet is to describe a sensible and reasonable sized functional and behavioral aspect of the subsystem, called a subsystem operation. The subsystem

operation specifies certain computations, calculations or assignments to be performed by the subsystem. A sequence of operations may exist where, a subsystem operation invokes another subsystem operation and so on and so forth. The Subsystem Operation Sheet does not describe, when or inside what sequence, the operation should be used.

The description of an operation should make use of the concepts defined in the structural model and vice versa. The states defined by statecharts in the dynamic model may also be included. If the operation is triggered by an event, it should also be stated. Any subsystem operation must be traceable via the responsibilities of the subsystem to the use cases.

The formality of the description may vary considerably. Pseudo-programming language statements or mathematical expressions are recommended. Figures or charts can also be used. Flow charts in particular, may be a good way to document complicated algorithms.

**Phase:** subsystem analysis

**Notation:** text table notation

**Mapping:** two columns; fixed number of rows; row headers in left column

**Template:**

|                       |   |
|-----------------------|---|
| <b>Operation</b>      | ID + operation name   |
| <b>Preconditions</b>  | conditions that need to be satisfied to start the operation, these do not guarantee that the operation will be successfully completed |
| <b>Description</b>    | short statement describing the operation  |
| <b>Inputs</b>         | arguments that an operation needs to perform its desired function   |
| <b>Modifies</b>       | what modifications the operation cause on its arguments or on common data in the subsystem  |
| <b>Outputs</b>        | what information the operation shall supply to others   |
| <b>Postconditions</b> | conditions after the operation is successfully completed and the conditions that apply if the operation is terminated due to an error |
| <b>Related To</b>     | associations to the classes and objects and possibly also to the events and states to which it is related                             |
| <b>Exceptions</b>     | Conditions that apply if the operation is terminated without successful completion  |

## 2.15 SUBSYSTEM PREEMPTION TABLE<sup>(A7)</sup>

The purpose of the Subsystem Preemption Table<sup>(A7)</sup> is to capture the decisions made on how the processes of a subsystem shall preempt each other if more than one process received an inter-process message.

**Phase:** subsystem design

**Notation:** text table notation

**Mapping:** floating number of columns and rows

**Template:**

| running | top-ready |    |    |    |
|---------|-----------|----|----|----|
|         | P1        | P2 | P3 | P4 |
| P1      | X         | Y  | Y  | N  |
| P2      | N         | X  | Y  | N  |
| P3      | N         | N  | X  | N  |
| P4      | N         | Y  | Y  | X  |

## 2.16 SYSTEM ARCHITECTURE DIAGRAM<sup>(A1)</sup>

The purpose of the System Architecture Diagram is to show the decomposition of the system into domains and subsystems. It also shows the static relationships between the subsystems. Each subsystem is developed further by a subsystem development team. Each subsystem development team executes its own subsystem development sequence, producing for each phase a specific set of artifacts.

The System Architecture Diagram can be a single one-page diagram or a set of linked diagrams. For example, it might be worthwhile to produce a diagram showing the decomposition only, and a separate set of diagrams showing the relationships between subsystems.

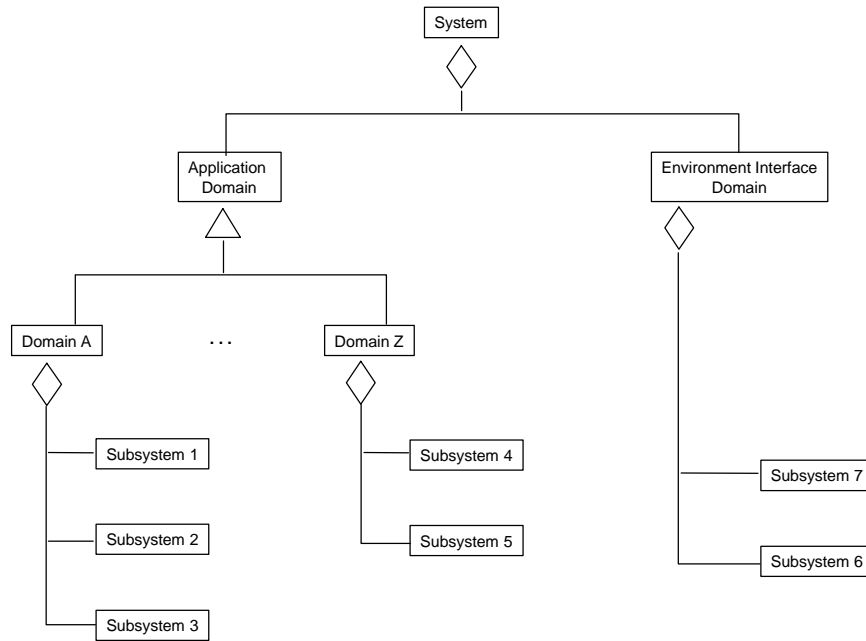
Optionally, the attachment<sup>(C27)</sup> Responsibility Matrix<sup>(D31)</sup> can be used in conjunction with the System Architecture Diagram.

**Phase:** system architecture

**Notation:** class diagram notation

**Mapping:** system, subsystem and domains as classes; decomposition of system as aggregation hierarchy; relationships between subsystems as associations between classes

**Template:**



**Attachment:** Responsibility Matrix<sup>(D31)</sup>

The purpose of the Responsibility Matrix is to document the mapping between use cases and subsystems. The subsystems must provide the use of the system as defined by use cases. Therefore, any use case, or part of it, can be assigned to a subsystem.

If a certain subsystem provides a use case, or part of it, a responsibility is defined. A unique ID for this responsibility is entered into the cell of the matrix where use case and subsystem match. An empty cell at subsystem X and use case Y indicates that subsystem X neglects use case Y.

**Notation:** text table

**Mapping:** as many columns as subsystems; as many rows as use cases;

**Template:**

|     | Subsystem 1       | Subsystem 2       | ... |
|-----|-------------------|-------------------|-----|
| U1  |                   |                   |     |
| U2  | Responsibility ID | Responsibility ID |     |
| ... |                   |                   |     |

**2.17 SYSTEM CONTEXT DIAGRAM**

The purpose of the System Context Diagram is to show structural information about the system and its actors. The system is represented as a black box, and the actors are located in the environment of the system. The actors explicitly request a use of the system or they are autonomously provided by some use of the system, described in the use cases.

The development work if started with the System Context Diagram artifact must cover and unfold the black box. The actors are out of the scope of this work.



the subsystem boundary, the Unqualified Event Thread stops with the first object outside. This may be an output event. The Unqualified Event Thread does not show conditions tested, computation or data flow.

The developer must produce as many Unqualified Event Threads as input events which are defined in the input event list. However, any number of unqualified event threads may be overlaid in a single figure if desirable and the result is nonetheless readable.

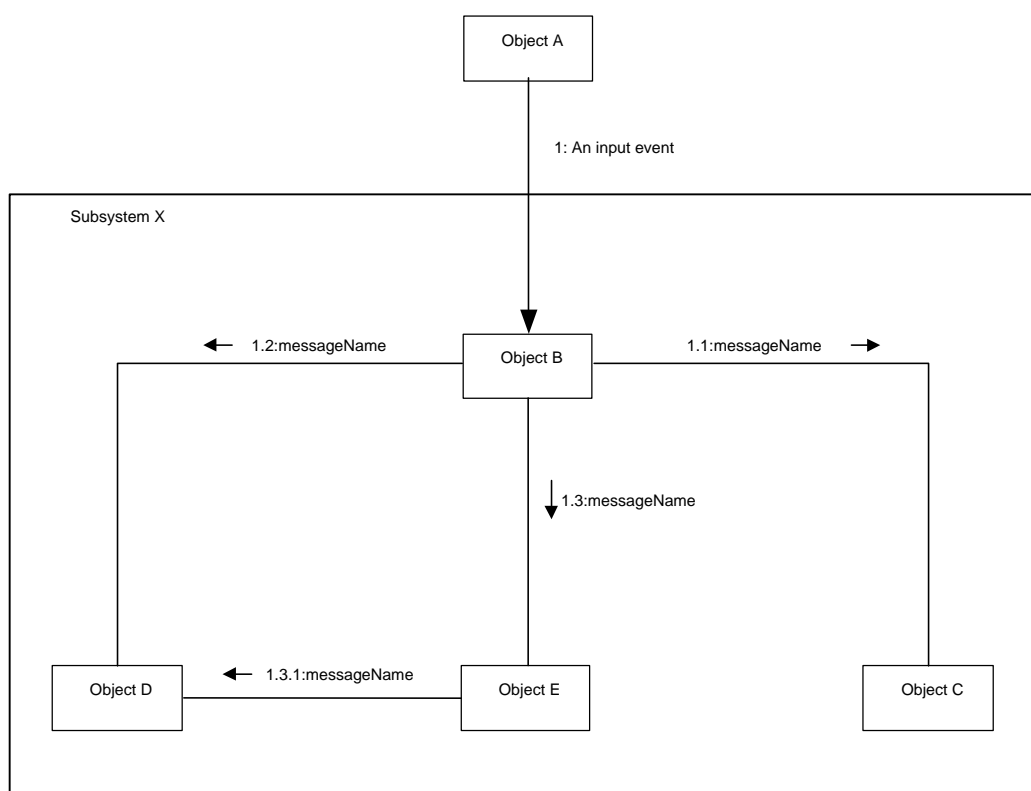
Note that the same design decisions are also captured in the Class Outlines and that all the details missing in the event thread are stated there. However, this information is distributed among the Class Outlines of the participating objects.

**Phase:** subsystem design

**Notation:** collaboration diagram with Octopus enhancements

**Mapping:** subsystem as composite object; objects of the subsystem inside this composite object; originator as object outside; event as defined by Octopus; interaction as collaboration with standard links and message flow (Octopus doesn't care about the type of little arrow)

**Template:**



## 2.19 USE CASE SHEET<sup>(B10)</sup>

The purpose of the Use Case Sheet is to describe a use case. The use case views the system as a black box and specifies a single usage of the system that has a purpose of reasonable scope and complexity.

A use case is either explicitly requested by an actor, or autonomously provided to an actor. The lifetime of an explicitly requested use case is determined by its actor. The lifetime of an autonomously provided use case is either infinitive, that is the lifetime of the system, or it is determined by explicitly requested use cases.

A use case can be either a sub-use case or super-use case. Developing such an hierarchy is recommended, if size or complexity of a use case gets unpractical, if parts of it require special attention or, if a number of use cases can be considered as parts of an embracing one. A sub-use case imports actors and preconditions and exports postconditions.

**Phase:** system requirements specification

**Notation:** text table

**Mapping:** two columns; fixed number of rows; row headers in left column

**Template:**

|                       |   |
|-----------------------|---|
| <b>Use Case</b>       | ID + use case name  |
| <b>Actors</b>         | Explicitly requests or is autonomously provided by some use of the system; external to the system as defined in the system context diagram; if sub use case, refer to actors of super use case  |
| <b>Preconditions</b>  | Conditions that need to be satisfied to perform the use case, these do not guarantee that the use case will be successfully completed; if sub use case, refer to ongoing super use case   |
| <b>Description</b>    | Statements describing what use of the system the actor requests or is provided to; if super use case, include references to sub use cases; if possible include time requirements  |
| <b>Exceptions</b>     | Statements describing anything possible to occur in the scope of this use case and not covered in the description field above; if important enough or complicated or otherwise worth, this description may be put in its own sheet; this new use case is a sub use case of this use case, but has the relation "exception" to it; insert reference here |
| <b>Postconditions</b> | Conditions after the use case has been successfully completed; if this sheet describes exceptions, also conditions after an exception has terminated the use case; if super use case, the postconditions of the sub use cases apply too   |