

# Release of Octopus/UML

Domiczi Endre, Farfarakis Rallis, Ziegler Jürgen  
Nokia Research Center

1.	INTRODUCTION .....	1
2.	THE OCTOPUS MISSION .....	2
3.	KEY CONCEPTS.....	2
3.1	Artifacts .....	2
3.2	Facilities .....	3
3.3	Development Sequence .....	3
3.4	Life-cycle Models.....	4
4.	CHANGE AND ENHANCEMENT REFERENCE.....	6
5.	BIBLIOGRAPHY.....	8

## 1. INTRODUCTION

Octopus/UML is the new enhanced version of the Octopus method [1]. The development of Octopus/UML is based on the real life experience gained since publishing the book describing Octopus [1] and is influenced by the appearance of the Unified Modeling Language (UML) [2], a de facto standard.

As any other method in the field of software development, Octopus/UML comprises techniques, which codify experience and knowledge in a form that can be used by the development team in order to help them proceed in a controlled and systematic manner. Octopus/UML, following the objectives of Octopus, departs from many other methods because it bridges the gap between object-oriented methods and real-time systems.

Octopus/UML includes the name UML because it complies with the terminology and notation defined by UML. Where appropriate it goes beyond UML, due to the special constraints of real-time systems. Despite the change in the name, compliance with UML demanded just a few changes on the former Octopus notation, since the authors of Octopus had anticipated the forthcoming UML notation. The major contributions are the introduction of new artifacts reflecting the experience gained in real-life usage, in combination with research and development done since the release of Octopus.

In order to make it possible to trace the changes and enhancements of Octopus/UML, those items are marked by superscripts when they appear the first time in a certain context. For more details refer to Section Change and Enhancement Reference.

Although Octopus/UML is highly UML compliant, a few adornments, called Special Octopus Features (SOF)<sup>(C28)</sup>, are not. Octopus/UML insists in those SOFs because they are required to express certain real-time issues, but UML is (so far!) incapable to do so. For more information refer to Chapter Octopus/UML Notation Summary.

## 2. THE OCTOPUS MISSION

Octopus/UML is an object-oriented software development method specialised for embedded real-time systems. A method specifies a framework in which work can be managed; identifying what should be done, when things should be done, what should be delivered and the language of deliverables. An object-oriented method, like Octopus/UML, follows the same principles and furthermore uses objects and classes as the principal focus of attention. The purpose of the method is to create models which elaborate on issues such as:

- Which classes and objects exist
- The structure, behaviour and purpose of those objects and classes
- The structure and dynamics of relationships between objects
- The structural relationships between classes.
- The smooth transition from objects and classes to tasks and processes.

The object-oriented method uses these models throughout the development phases; from requirements capture, analysis and design, implementation, to testing and maintenance. A model represents an abstraction of a problem in order to assure understanding prior to its development. Each model covers what artifacts to produce, why they exist, how they are structured in relation to each other and how they relate dynamically. Artifacts are essential for team communication and common understanding. Having a common language to communicate and criticise ideas is definitely the most important aspect of a development method. The visualisation of what a system can do, before coding, can substantially increase the quality of systems and code. Producing artifacts is the essence of Octopus/UML, as well as Octopus.

Object-oriented technology is simply part of the evolution, and not the revolution, of development technology and techniques. The technology has evolved and is being used because it works both in the short and in the long term. In the short term, object technology provides a direct mapping from object-oriented concepts, models and artifacts to object-oriented programming languages such as C++ and Smalltalk. In the long term, organisations build up libraries of components which can be reused in new applications, providing more rapid development and faster investment return.

## 3. KEY CONCEPTS

### 3.1 ARTIFACTS

An artifact is termed as an “*man-made object*” [6]. In the context of Octopus/UML, an artifact describes a sensible and valuable piece of information, consisting of an aim, the semantics, a supporting notation and a template.

Software developers applying Octopus/UML are guided to produce a number of well-defined artifacts. With the help of these artifacts the developer captures, expresses and communicates essential development results in concrete terms. For an example see Figure 3-1.

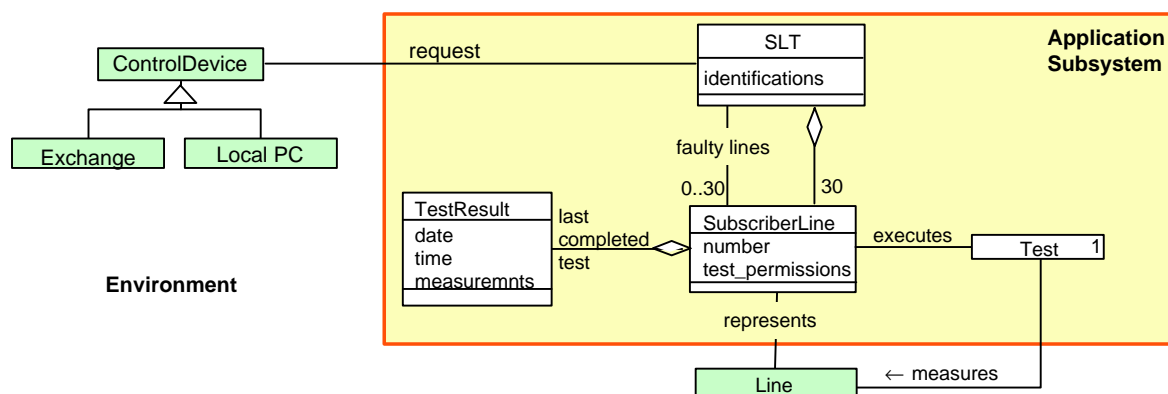


Figure 3-1. An Analysis Class Diagram artifact.

Each artifact is defined for an individual purpose. Production of an artifact contributes a sensible and valuable piece of information to the development progress. However, looking at individual artifacts in isolation is not intended, since, the real value stems from using the artifacts in collaboration. The set of artifacts, the guidelines to produce one, the links from one to another and the recommended order to proceed, provides the means for an effective development progress.

To produce a certain artifact you must follow the guidelines. Each artifact uses at least one notation that must be applied to produce it. Some artifacts can be produced by applying one out of an alternative list of notations. The same notation can be used for many artifacts. Wherever feasible, the notation suggested is the UML notation. Please notice the strict difference made in Octopus/UML between the artifact and the notation applied.

Some artifacts have an optional attachment<sup>(C27)</sup>. An attachment is separate from the artifact, but is closely linked to it. It is used in cases where it is advisable to explain or describe some items of the artifact in more detail.

The properties of each artifact are defined in detail in the Chapter Octopus/UML Artifact Reference. Any graphical notation these artifacts refer to is listed in the Chapter Octopus/UML Notation Summary.

### 3.2 FACILITIES

Besides artifacts, Octopus/UML defines facilities<sup>(C26)</sup>. Facilities are optional work products similar to artifacts, however their content is of secondary importance. They can be used to help developers organise and manage more efficiently their work. In situations where the information is quite complex, facilities assist in reducing ambiguities and inconsistencies. For more information about the suggested facilities refer to the Chapter Octopus/UML Facility Reference.

### 3.3 DEVELOPMENT SEQUENCE

All artifacts are linked with each other. However, in order to achieve a specific goal at a certain stage during development, more than one artifact may be required. These artifacts are clearly more closely related to each other than others. The development sequence codifies this by defining phases and models that structure and guide the progress of development work based on the artifacts.

A phase requests to build at least one model. Each model defines a specific set of artifacts. The artifacts of a model are most related to each other. See Figure 3-2 for an example. Only when browsing the complete set of artifacts, you will understand the model. When developing them, you must progress on all of them in a balanced speed. If a phase defines several models, the artifacts of different models are less related than inside a model. In some phases the models must be developed in parallel, in other ones partial serialisation is possible. Ideally, phases are in series.

It is important to note that Octopus/UML does not suggest a fixed sequence that all projects must follow. Instead only a pattern is given, following which each project is invited to configure its own sequence.

The development sequence, its generic pattern and each phase including the models to be built are defined in detail in the Chapter Octopus/UML Sequence Reference.

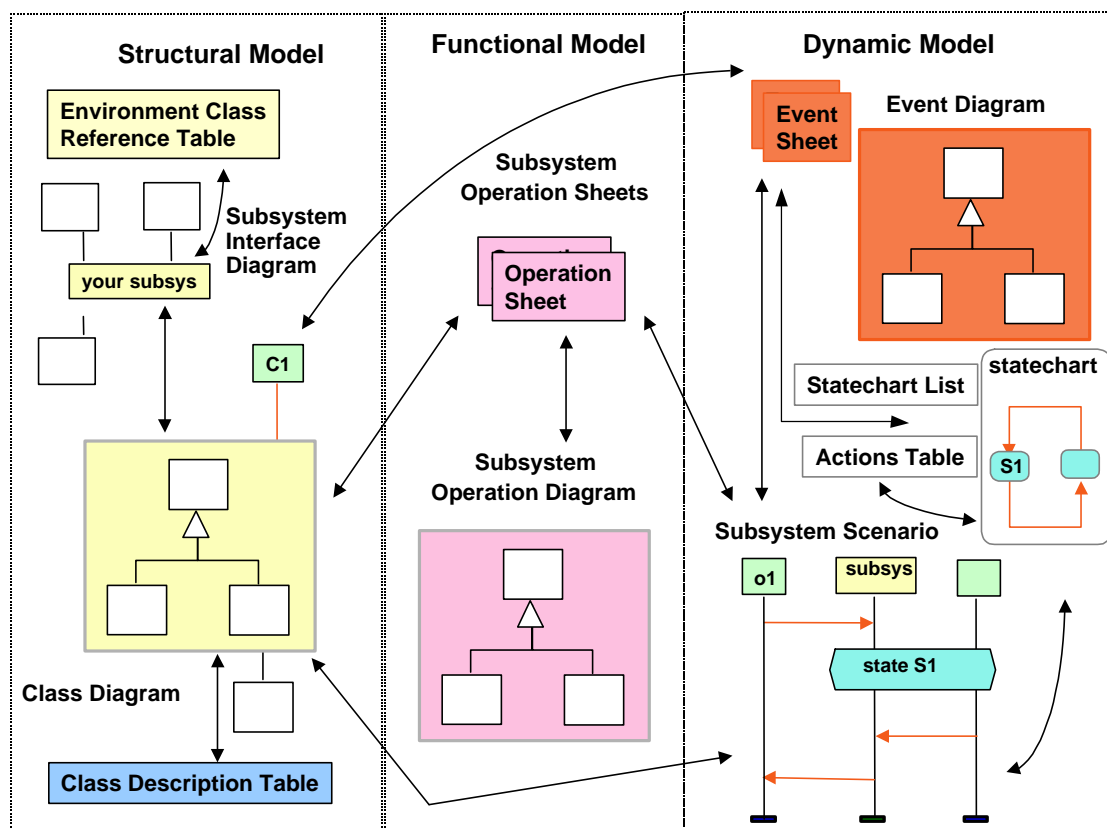


Figure 3-2. Models and Artifacts of Subsystem Analysis Phase.

### 3.4 LIFE-CYCLE MODELS

Two object-oriented software life-cycle models exist: the *incremental* and the *evolutionary* model. Octopus/UML allows any combination of the two models because real-life projects desire such flexibility. However, the principles of each should be understood separately.

In the incremental model [3, 4, 5], user needs are determined and system requirements are defined first. The rest of the development is then performed in a sequence of increments (Figure 3-3). The first increment incorporates part of the planned capabilities, the next increment adds more capabilities, and so on, until the system is complete. Although user needs and system requirements should be well defined beforehand, the advantage is that the increments are much more easily to develop.

In the evolutionary model [3, 4, 5], a system is also developed in increments, but it differs from the incremental model firstly, in acknowledging that the user needs are not fully understood and secondly, in that all of the requirements cannot be defined up front. User needs and system requirements are partially defined in the beginning and updated in each succeeding increment. The evolutionary model does not assume that it will get the system right the first time. Rather, the developers work on an increment and the results are fed back to the development work of the next increment (Figure 3-4). Successive versions of the system are built, until the final system emerges.

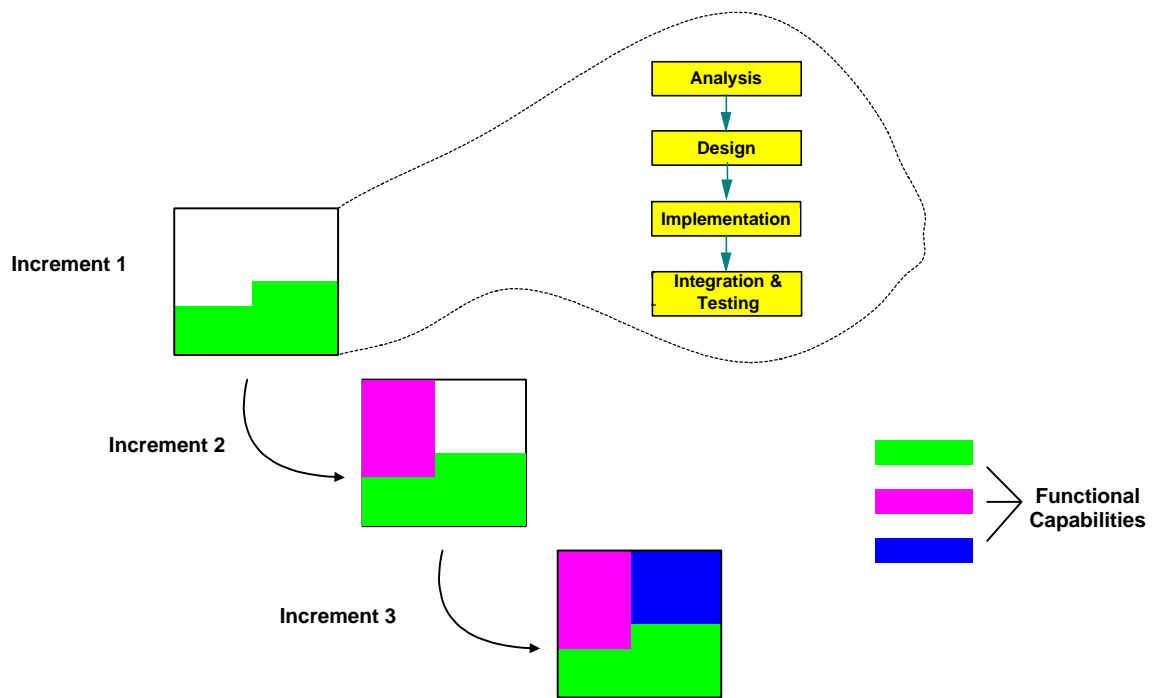


Figure 3-3. Incremental Model.

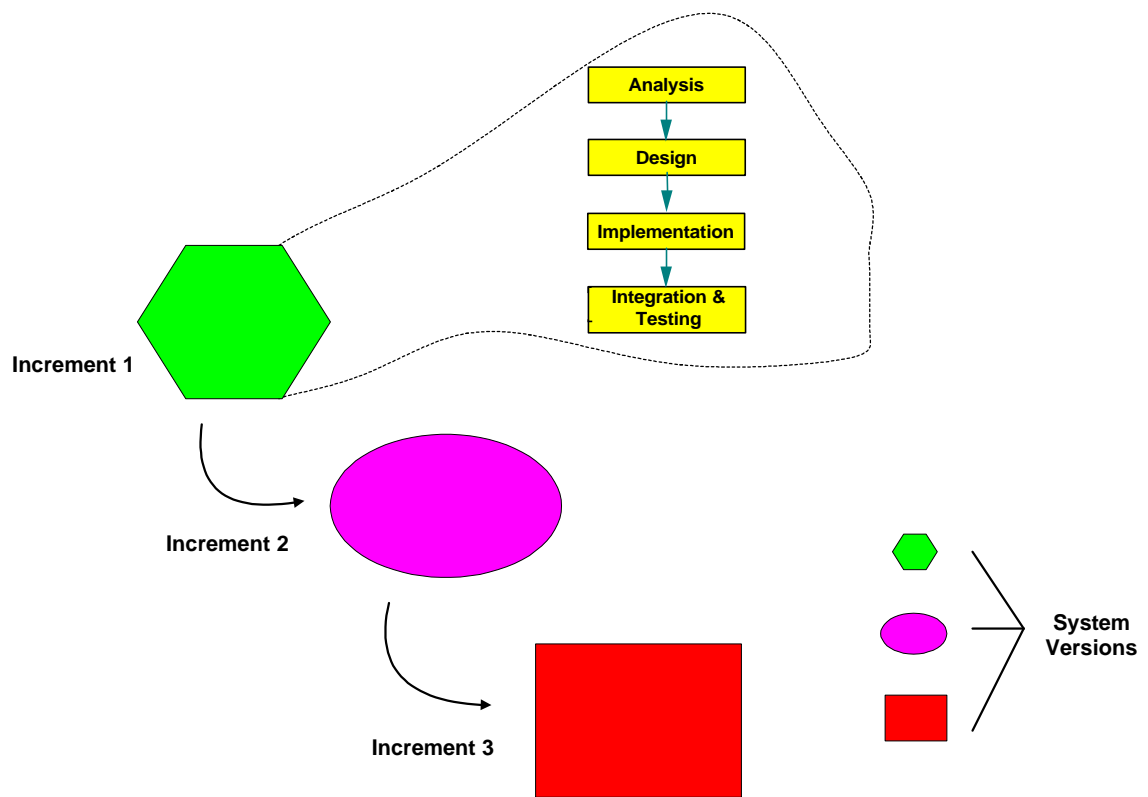


Figure 3-4. Evolutionary Model.

#### 4. CHANGE AND ENHANCEMENT REFERENCE

This section presents the changes and enhancements proposed by Octopus/UML in relation to the contents of the book [1]. The changes and enhancements are recorded in Table 4-1, which the reader can use as a reference tool.

A unique alphanumeric identification (ID) has been assigned to every change or enhancement. The IDs of the table appear as superscripts (e.g. ...Event Diagram<sup>(A6)</sup>...) in the following chapters of this Octopus Supplement Volume I. The reader may refer to the table below for clarification whenever such a superscript is seen in the text. A brief description on the clarification of a change or enhancement is also included for every entry in the table.

The Octopus/UML changes and enhancements are classified according to four different groups:

- A. *Renamed item.* This group refers to items that already existed in the book and now they have been renamed. Items may include artifacts, reserved keywords or concepts.
- B. *Representation issue.* This group records the changes or enhancements related to the representation of the material found in the book.
- C. *New or revised term/concept.* The elements of this group are changes or enhancements on the meaning of terms or concepts that exist in the book.
- D. *New artifact/facility/attachment.* The last group presents the new artifacts, facilities<sup>(C26)</sup> or attachments<sup>(C27)</sup> that have been developed in Octopus/UML to serve the needs of the users.

Table 4-1. Changes and Enhancements compared to Octopus.

ID	Change or Enhancement	Description
<b>Renamed item</b>		
A1	System Architecture Diagram	Formerly known as Subsystems Diagram.
A2	Subsystem Operation Sheet	Formerly known as Operation Sheet.
A3	Input Event Sheet	Formerly known as Event Sheet.
A4	Collaboration Diagram	Formerly known as Object Interaction Graph.
A5	Analysis Class Diagram	Formerly known as Class Diagram.
A6	Event Diagram	Formerly known as Event Grouping Diagram.
A7	Subsystem Preemption Table	Formerly known as Preemption Table.
A8	Subsystem Scenario	Formerly known as Scenario.
A9	Analysis Class Description Table	Formerly known as Class Description Table.
A10	Sequence Diagram	Formerly known as Message Sequence Chart.
A11	Inter-Process Message Outline	Formerly known as Message Outline.
<b>Representation issue</b>		
B10	Use case sheet template	Rearrangement and renaming of fields in the template.
B11	Subsystem operation sheet template	Rearrangement and renaming of fields in the template.
B12	Octopus enhancement	Represents an enhancement to Octopus notation that is not related to UML whatsoever. See in Octopus/UML Notation Reference.
B13	Octopus/UML enhancement	Represents an enhancement to Octopus notation that is a result of combined Octopus/UML ideas.

B14	Event sheet template	Rearrangement and renaming of fields in the template.
B15	Process Function Outline	Addition of the SDL/GR process diagram notation as an alternative in the description of processes.
B16	Class Outline	Addition of the SDL/GR process diagram notation as an alternative in the description of class member functions.
<b>New or revised term/concept</b>		
C21	Event	An event was always qualified by an asynchronous mechanism. Now, an event is always unqualified and when qualified, it is changed to a member function call or an inter-process message.
C22	Environment wrapper	Replaces the term hardware wrapper. It refers to a broader view that the wrapper takes towards the system. Besides hardware-related concepts, the environment wrapper takes into account software-related issues such as the handling of legacy cod.
C23	Input/output event	The term event always referred to an input event. Now there is a distinction on input and output event of a subsystem.
C24	Primary/secondary event	This is a new distinction based on the source of the event. Output events of the environment wrapper are termed as primary events. Input or output events between application domain subsystems are generally termed as secondary events.
C25	Primitive event	Events that originate from the hardware and target the environment wrapper are termed as primitive events.
C26	Facility	Similar to artifacts but of secondary importance.
C27	Attachment	Explains some items of an artifact.
C28	Special Octopus Feature (SOF)	A special adornment that represents real-time issues.
<b>New artifact/facility/attachment</b>		
D23	System Scenario	Shows the interactions of the system with the actors. It is included in system requirements specification phase.
D24	Responsibility Sheet	Lists the responsibilities of a discrete subsystem. It belongs to the system architecture phase.
D25	Inter-Subsystem Scenario	Presents the interactions of the subsystems and actors. It is added in the system architecture phase.
D26	Design Class Description Table	Presents information about the design classes, including data members and member functions. It is added in the subsystem design phase.
D28	Subsystem Inter-Process Communication Diagram	Provides a top level graphical view of the processes of a subsystem and their communication. It is added in the subsystem design phase.
D30	Subsystem Interface Diagram	Shows the static view of the subsystem's interface with neighbouring subsystems. It belongs to the subsystem analysis phase.
D31	Responsibility Matrix	Shows the relationships and responsibilities between the use cases and the subsystems. It is added in the system architecture phase.
D32	Subsystem Inter-Process Scenario	Presents the message flow between processes. It is added in the subsystem design phase.

D33	Design Class Diagram	Shows the static view of design classes and their relationships. It is added in the subsystem design phase.
D34	Inter-Process Message List	A list that identifies and describes all inter-process messages. It is part of the subsystem design phase.
D35	Subsystem Operation Diagram	Regarded as a roadmap of the subsystem operations identified in the functional model. It belongs to the subsystem analysis phase.

## 5. BIBLIOGRAPHY

- [1] M. Awad, J. Kuusela and J. Ziegler, *Object-Oriented Technology for Real-Time Systems*. Prentice Hall, 1996.
- [2] Unified Modeling Language, Edition 1.1, January 1997.
- [3] A. T. F. Hutt, *Object Analysis and Design. Comparison of Methods*, The Object Management Group, John Wiley & Sons, 1994.
- [4] J. Sodhi and P. Sodhi, *Object-Oriented Methods for Software Development*, McGraw Hill, 1996.
- [5] T. Cotton, *Evolutionary Fusion: A Customer-oriented Incremental Life Cycle for Fusion*, Hewlett-Packard Journal, August 1996.
- [6] Longman Dictionary of Contemporary English, Longman Group Ltd., 1980.