ISO/IEC 19793

**Date:** 2005-12-12

**WG19 Helsinki 2005**

**Committee Draft v02.00**

**Output Document from the May 2005 meeting in Helsinki**

**ISO/IEC JTC 1/SC 7**

**Software Engineering**

**Secretariat: Canada (SCC)**

| | |
|---|---|
| **Doc Type:** | Committee Draft |
| **Title:** | Information technology — Open distributed processing —<br>Use of UML for ODP system specifications<br>ITU-T Recommendation X.906 \| ISO/IEC 19793 |
| **Source:** | WG19 output |
| **Project:** | 1.07.19793 |
| **Status:** | Consolidated Input for the progression of CD v2.0 of ITU-T Recommendation X.906 \| ISO/IEC 19793 |
| **Action:** | |
| **Distribution:** | SC 7/WG19 and ITU-T |
| **Medium:** | E |
| **Number of Pages:** | |
| **Version:** | 02.00 |

ISO/IEC JTC1/SC7 Secretariat
École de Technologie Supérieure – ETS
1100 Notre-Dame Ouest,
Montréal, Québec
Canada H3C 1K3
Phone: +1 - 514 - 396-8632
Fax: +1 - 514 - 396-8684

## CONTENTS

# Foreword

This is Committee Draft version 02.00 of ITU-T Recommendation X.906 | ISO/IEC International Standard 19793: *Information technology — Open distributed processing — Use of UML for ODP system specifications*. It is the output from the May 2005 meeting of SC7 WG19 in Helsinki which continued in Bari in October 2005.

# 0 Introduction

The rapid growth of distributed processing has led to the adoption of the Reference Model of Open Distributed Processing (RM-ODP). This Reference Model provides a co-ordinating framework for the standardisation of open distributed processing (ODP). It creates an architecture within which support of distribution, interworking, and portability can be integrated. This architecture provides a framework for the specification of ODP systems.

RM-ODP is based on precise concepts derived from current distributed processing developments and, as far as possible, on the use of formal description techniques for specification of the architecture. It does not recommend any notation.

The Unified Modelling Language (UML) was developed by the Object Management Group (OMG). It provides a notation for modelling in support of information system design and is widely used throughout the IT industry as the language and notation of choice.

This Recommendation | International Standard refines and extends the definition of how ODP systems are specified by defining the use of the Unified Modelling Language for the expression of ODP system specification.

## 0.1 RM-ODP

The RM-ODP consists of:

– ITU-T Recommendation X.901 | ISO/IEC 10746-1: Overview, which contains a motivational overview of ODP, giving scoping, justification and explanation of key concepts, and an outline of the ODP architecture. It contains explanatory material on how the RM-ODP is to be interpreted and applied by its users, who may include standards writers and architects of ODP systems. It also contains a categorisation of required areas of standardisation expressed in terms of the reference points for conformance identified in ITU-T Recommendation X.903 | ISO/IEC 10746-3. This part is not normative.

– ITU-T Recommendation X.902 | ISO/IEC 10746-2: Foundations, which contains the definition of the concepts and analytical framework for normalised description of (arbitrary) distributed processing systems. It introduces the principles of conformance to ODP standards and the way in which they are applied. This is only to a level of detail sufficient to support ITU-T Recommendation X.903 | ISO/IEC 10746-3 and to establish requirements for new specification techniques. This part is normative.

– ITU-T Recommendation X.903 | ISO/IEC 10746-3: Architecture, which contains the specification of the required characteristics that qualify distributed processing as open. These are the constraints to which ODP standards shall conform. It uses the descriptive techniques from ITU-T Recommendation X.902 | ISO/IEC 10746-2. This part is normative.

– ITU-T Recommendation X.904 | ISO/IEC 10746-4: Architectural semantics, which contains a formalisation of the ODP modelling concepts defined in clauses 8 and 9 of ITU-T Recommendation X.902 | ISO/IEC 10746-2. The formalisation is achieved by interpreting each concept in terms of the constructs of one or more of the different standardised formal description techniques. This part is normative.

In the same series as the RM-ODP are a number of other standards and recommendations, and, of these, the chief that concerns this Recommendation | International Standard is:

– ITU-T Recommendation X.911 | ISO/IEC 15414: Enterprise language, which refines and extends the enterprise language defined in ITU-T Recommendation X.903 | ISO/IEC 10746-3 to enable full enterprise viewpoint specification of an ODP system.

## 0.2 UML

The Unified Modelling Language (UML) is a visual language for specifying and documenting the artefacts of systems. It is a general-purpose modelling language that can be used with all major object and component methods and that can be applied to all application domains (e.g., health, finance, telecom, aerospace) and implementation platforms (e.g., J2EE, CORBA, .NET). UML 2.0 has been structured modularly, with the ability to select only those parts of the language that are of direct interest. It is extensible, so it can be easily tailored to meet the specific user requirements.

UML defines twelve types of diagrams, divided in three categories: static application structure; dynamic behaviour; and organization and management of the application's modules. In addition, UML incorporates powerful extensions mechanisms that allow the definition of new dialects of UML to customize the language for particular platforms and domains.

The UML specification is defined using a metamodelling approach (i.e., a metamodel is used to specify the model that comprises UML). That metamodel has been architected so that the resulting family of UML languages is fully aligned with the rest of the OMG specifications (e.g., MOF, OCL, XMI) and initiatives (e.g., MDA), and to allow the exchange of models between tools.

## 0.3 Overview and motivation

ITU-T Recommendation X.903 | ISO/IEC 10746-3 defines a framework for the specification of ODP systems comprising

a) five viewpoints, called enterprise, information, computational, engineering and technology, which provide a basis for the specification of ODP systems;

b) a viewpoint language for each viewpoint, defining concepts and rules for specifying ODP systems from the corresponding viewpoint.

This Recommendation | International Standard defines:

– use of the viewpoints prescribed by the RM-ODP to structure UML system specifications;

– rules for expressing RM-ODP viewpoint languages and specifications with UML and UML extensions (e.g. UML profiles).

It allows UML tools to be used to process viewpoint specifications, facilitating the software design process.

Currently there is growing interest in the use of UML for system modelling. However, there is no widely agreed approach to the structuring of such specifications. This adds to the cost of adopting the use of UML for system specification, hampers communication between system developers and makes it difficult to relate or merge system specifications where there is a need to integrate IT systems.

The RM-ODP family of recommendations and international standards defines essential concepts necessary to specify open distributed processing systems from five prescribed viewpoints and provides a well-developed framework for the structuring of specifications for large-scale, distributed systems.

However, the RM-ODP family of standards is notation free, as well as model development method free. This document defines a notation for the ODP system specification concepts and structuring approaches for system specification using the notation, thus providing the basis for model development methods.

By defining how UML and UML extensions should be used to represent RM-ODP viewpoint languages and express viewpoint specifications, the standard enables the ODP viewpoints and ODP architecture to provide the needed framework for system specification using UML.

This Recommendation | International Standard contains the following annexes:

– Annex A: Summary of UML profiles of ODP languages using ITU-T guidelines for UML profile design

– Annex B: Example specifications

– Annex C: Relationship with MDA®

– Annex D: Architectural styles

Annexes B, C and D are not normative.

**INTERNATIONAL STANDARD**
**ITU-T RECOMMENDATION**

# Information technology — Open distributed processing — Use of UML for ODP system specifications

## 1 Scope

This Recommendation | International Standard defines use of the Unified Modelling Language (UML, OMG documents ptc/03-12-01, UML 2.0 Infrastructure Specification, and formal/05-04-07, UML 2.0 Superstructure Specification) for expressing system specifications in terms of the viewpoint specifications defined by the Reference Model of Open Distributed Processing (RM-ODP, ITU-T Rec. X.901 to X.904 | ISO/IEC 10746 Parts 1 to 4) and the Enterprise Language (ITU-T Rec. X.911 | ISO/IEC 15414). It covers:

a) the expression of a system specification in terms of RM-ODP viewpoint specifications using defined UML concepts and extensions (e.g. structuring rules, technology mappings, etc.);

b) relationships between the resultant RM-ODP viewpoint specifications;

c) relationships between RM-ODP viewpoint specifications and model driven architectures such as the OMG MDA.

This document is intended for the following audiences:

– ODP modellers who want to use the UML notation for expressing their ODP specifications in a graphical and standard way;

– UML modellers who want to use the RM-ODP concepts and mechanisms to structure their UML system specifications; and

– modelling tool suppliers, who wish to develop UML-based tools that are capable of expressing RM-ODP viewpoint specifications.

## 2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

### 2.1 Identical Recommendations | International Standards

– ITU-T Recommendation X.902 (1995) | ISO/IEC 10746-2:1996, *Information technology – Open Distributed Processing – Reference Model: Foundations*.

– ITU-T Recommendation X.903 (1995) | ISO/IEC 10746-3:1996, *Information technology – Open Distributed Processing – Reference Model: Architecture*.

– ITU-T Recommendation X.904 (1997) | ISO/IEC 10746-4:1998, *Information technology – Open Distributed Processing – Reference Model: Architectural semantics*.

– ITU-T Recommendation X.911 (2002) | ISO/IEC 15414:2002, *Information technology – Open distributed processing – Reference model – Enterprise language*.

– ITU-T Recommendation X.725 | ISO/IEC 10165-7, *Information Technology – Open Systems Interconnection – Structure of Management Information – Part 7: General Relationship Model*

### 2.2 OMG specifications

- OMG document ptc/04-10-14, *UML 2.0 Infrastructure Specification*
- OMG document formal/05-04-07, *UML 2.0 Superstructure Specification*

## 3 Definitions

For the purposes of this Recommendation | International Standard, the following definitions apply.

### 3.1 Definitions from ODP standards

#### 3.1.1 Modelling concept definitions

This Recommendation | International Standard makes use of the following terms as defined in ITU-T X.902 | ISO/IEC 10746-2:

abstraction; action; activity; architecture; atomicity; behaviour (of an object); binding; class; client object; communication; composition; component object [2-5.1]; composite object; configuration (of objects); conformance point; consumer object; contract; creation; data; decomposition; deletion; distributed processing; distribution transparency; <X> domain; entity; environment; environment contract; epoch; error; establishing behaviour; failure; fault; <X> group; identifier; information; initiating object; instance; instantiation (of an <X> template); internal action; interaction; interchange reference point; interface; interface signature; interworking reference point; introduction; invariant;location in space; location in time; name; naming context; naming domain; notification; object; obligation; ODP standards; ODP system; open distributed processing; perceptual reference point; permission; persistence; producer object; programmatic reference point; prohibition; proposition; quality of service; reference point; refinement; role; server object; spawn action; stability; state (of an object); subdomain; subtype; supertype; system; <X> template; term; terminating behaviour; trading; type (of an <X>); viewpoint (on a system).

#### 3.1.2 Viewpoint language definitions

This Recommendation | International Standard makes use of the following terms as defined in ITU-T X.903 | ISO/IEC 10746-3:

binder; capsule; channel; cluster; community; computational behaviour; computational binding object; computational object; computational interface; computational viewpoint; dynamic schema; engineering viewpoint; enterprise object; enterprise viewpoint; <X> federation; information object; information viewpoint; interceptor; invariant schema; node; nucleus; operation; protocol object; static schema; stream; stub; technology viewpoint; <viewpoint> language.

### 3.2 Definitions from the Enterprise Language

This Recommendation | International Standard makes use of the following terms as defined in ITU-T X.911 | ISO/IEC 15414:

actor (with respect to an action); agent; artefact (with respect to an action); authorization; commitment; community object; declaration; delegation; evaluation; field of application (of a specification); interface role; objective (of an <X>); party; policy; prescription; principal; process; resource (with respect to an action); scope (of a system); step; violation.

### 3.3 Definitions from the Unified Modelling Language

This Recommendation | International Standard makes use of the following terms as defined in OMG documents ptc/03-12-01 and formal/05-04-07:

abstract class; action; activity; activity diagram; aggregate; aggregation; association; association class; association end; attribute; behaviour; behaviour diagram; binary association; binding ; cardinality; call; class; classifier; classification; class diagram; client; collaboration; collaboration occurrence; communication diagram; component; component diagram; composite; composite structure diagram; composition; concrete class; connector; constraint; container; context; delegation; dependency; deployment diagram; derived element; diagram; distribution unit; dynamic classification; element; entry action; enumeration; event; exception; execution occurrence; exit action; export; expression;

extend; extension; feature; final state; fire; generalizable element; generalization; guard condition; implementation; implementation class; implementation inheritance; import; include; inheritance; initial state; instance; interaction; interaction diagram; interaction overview diagram; interface; internal transition; lifeline; link; link end; message; metaclass; metamodel; method; model element; multiple classification; multiplicity; n-ary association; name; namespace; node; note; object; object diagram; object flow state; object lifeline; operation; package; parameter; parameterized element; parent; part; partition; pattern; persistent object; pin; port; postcondition; precondition; primitive type; profile; property; pseudo-state; realization; receive [a message]; receiver; reception; refinement; relationship; role; scenario; send [a message]; sender; sequence diagram; signal; signature; slot; state; state machine diagram; state machine; static classification; stereotype; stimulus; structural feature; structure diagram; subactivity state; subclass; submachine state; substate; subpackage; subsystem; subtype; superclass; supertype; supplier; synch state; tagged value; time event; time expression; timing diagram; trace; transient object; transition; type; usage; use case; use case diagram; value; vertex; visibility.

### 3.4 Definitions from ODP standards refined or extended in this standard

This Recommendation | International Standard refines or extends the following terms from ITU-T X.902 | ISO/IEC 10746-2, ITU-T X.903 | ISO/IEC 10746-3, or ITU-T X.911 | ISO/IEC 15414:

Temporary Note – This clause will be deleted or completed when Clauses 7 to 11 are finalised.

## 4 Abbreviations

For the purposes of this Recommendation | International Standard, the following abbreviations apply.

| | |
|---|---|
| MDA® | Model Driven Architecture |
| ODP | Open Distributed Processing |
| RM-ODP | Reference Model of Open Distributed Processing |
| UML | Unified Modelling Language |

## 5 Conventions

In the following, this Recommendation | International Standard will be referred to as “this document”.

ITU-T Recommendation X.902 | ISO/IEC 10746-2 (RM-ODP Part 2: Foundations) and ITU-T Recommendation X.903 | ISO/IEC 10746-3 (RM-ODP Part 3: Architecture) are referred to as “Part 2” and “Part 3” of the RM-ODP, respectively.

ITU-T Recommendation X.911 | ISO/IEC 15414 (RM-ODP Enterprise Language) are referred to as “the Enterprise Language”.

OMG document formal/05-04-07 is referred as “the UML standard”

References to the normative text of this document, to the text of Parts 2 and 3 of the RM-ODP, and to the Enterprise Language are expressed in one of these forms:

| | |
|---|---|
| [Part 2 – n.n] | – a reference to clause n.n of RM-ODP Part 2; |
| [Part 3 – n.n] | – a reference to clause n.n of RM-ODP Part 3; |
| [E/L – n.n] | – a reference to clause n.n of the Enterprise Language; |
| [UML – n.n] | – a reference to clause n.n of the UML standard; |
| [n.n] | – a reference to clause n.n of this document. |

For example, [Part 2 – 9.4] is a reference to subclause 9.4 of Part 2 of the RM-ODP; and [6.5] is a reference to clause 6.5 of this document. These references are for the convenience of the reader.

NOTE – The clauses correspond to the specific dated versions of the documents referenced in Clause 2.

In the clauses that follow, except in the headings, terms in *italic* face are terms of the RM-ODP viewpoint languages as defined in Parts 2 and 3 of the RM-ODP, or in the Enterprise Language. UML concepts are shown in sans-serif typeface. UML stereotype names are shown in normal font, enclosed in guillemets (« and »).

The following conventions will apply to the UML diagrams:

- Association end names will be placed at the end of the association that is adjacent to the class playing the role. Association end names are omitted if they do not add meaning to the diagram. In this case, the implied association end name is the name of the class at that end of the association.
- Cardinalities of associations are placed adjacent to the class that has the cardinality.
- Where there are no attributes, the attribute part of the class box is suppressed.
- Black diamonds are used to represent whole/part associations, with no cardinality or role name at the whole end of the association, and no role name at the part end of the association. The meaning is that the part cannot exist without exactly one instance of the whole.
- The use of UML aggregation associations (i.e., those that use white diamonds) is discouraged in the diagrams.
- Nouns are used in association end names, rather than verbs.
- Class names representing ODP concepts start with upper case.
- Arrowheads accompanying association names are avoided.

# 6 Overview of modelling and system specification approach

## 6.1 Introduction

This clause provides an introduction to this document, covering:

- an overview of ODP system specification concepts;
- an overview of UML modelling concepts;
- an introduction to the approach taken in expressing ODP system specifications using UML;
- an overview of the structuring principles for system specifications defined in the document;
- an explanation of the concept of correspondences (relationships) between viewpoint specifications and how these are expressed using UML.

## 6.2 Universes of discourse, ODP specifications and UML models

In using the techniques described in this document, it is necessary to understand the relationships between the subject of a model, i.e., its Universe of Discourse (UOD), ODP specifications for that UOD and how those ODP specifications are represented in UML.

The four main sets of notions involved in understanding these relationships are:

- the entities, and the relationships amongst them, in the UOD being modelled;
- the ODP specification(s) of that UOD;
- the UML model(s) that represent the ODP specifications;
- the UML notation (diagramming techniques and other mechanisms) by means of which the UML models are expressed.

There are three important kinds of relationship between these notions.

- First, in the same way that an ODP object is defined as a model of an entity (a concrete or abstract thing of interest), an ODP specification is a model of a UOD. The modeller uses the concepts and structuring rules of RM-ODP Part 2, together with those of the relevant ODP viewpoint language(s) (RM-ODP Part 3 and the Enterprise Language), to produce a specification that represents relevant facts and assertions about the entities that exist in the UOD. The rules for this kind of relationship are stated in Parts 2 and 3 of the RM-ODP, and in the Enterprise Language.
- Secondly, instances of ODP viewpoint language concepts in the ODP specifications are represented by instances of one or more UML metaclasses that, through the relevant profile (set of stereotypes, tag definitions and constraints), map to and from the ODP concepts, to produce a UML model of the ODP specification. The rules for this kind of relationship are stated in this document.
- Thirdly, the UML notation is used to express graphically the underlying UML model. The rules for this kind of relationship are stated in the UML specification.

This document addresses the three simple relationships described above. While there are other derived relationships between elements in this chain (e.g., between UOD and UML model), they are not otherwise referred to in this document. These relationships are illustrated in Figure 1.

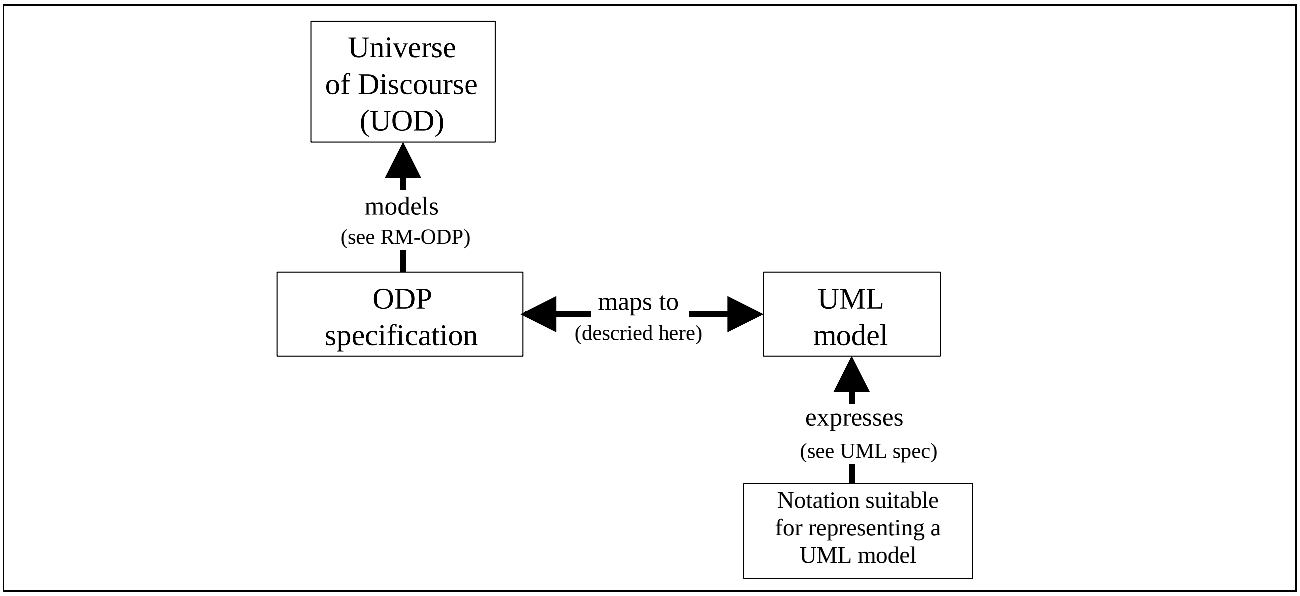



**Figure 1 – Relationships between UOD, ODP specifications, and UML models**

## 6.3 Overview of ODP concepts (extracted from RM-ODP Part 1)

An overview of the ODP modelling concepts and the structuring rules for their use is given in RM-ODP Part 1 (ITU-T Rec. X.901 | ISO/IEC 10746-1: Overview) and the concepts and structuring rules are formally defined in RM-ODP Parts 2 and 3. The text that follows (i.e. the rest of [6.3]), is abstracted from the text in RM-ODP Part 1. RM-ODP Parts 2 and 3 are the authoritative standards, and should be followed in case of any conflict between those Parts and this clause.

The framework for system specification provided by the RM-ODP has four fundamental elements:

- an object modelling approach to system specification;
- the specification of a system in terms of separate but interrelated viewpoint specifications;
- the definition of a system infrastructure providing distribution transparencies for system applications;
- a framework for assessing system conformance.

### 6.3.1 Object Modelling

Object modelling provides a formalization of well-established design practices of abstraction and encapsulation.

- Abstraction allows the description of system functionality to be separated from details of system implementation.
- Encapsulation allows the hiding of heterogeneity, the localization of failure, the implementation of security and the hiding of the mechanisms of service provision from the service user.

The object modelling concepts cover:

- basic modelling concepts: providing rigorous definitions of a minimum set of concepts (action, object, interaction and interface) that form the basis for ODP system descriptions and are applicable in all viewpoints;
- specification concepts: addressing notions such as type and class that are necessary for reasoning about specifications and the relations between specifications, providing general tools for design, and establishing requirements on specification languages;

– structuring concepts: building on the basic modelling concepts and the specification concepts to address recurrent structures in distributed systems, and covering such concerns as policy, naming, behaviour, dependability and communication.

### 6.3.2 Viewpoint specifications

A *viewpoint* (*on a system*) is an abstraction that yields a specification of the whole system related to a particular set of concerns. Five *viewpoints* have been chosen to be both simple and complete, covering all the domains of architectural design. These five viewpoints are:

– the *enterprise* viewpoint, which is concerned with the purpose, scope and policies governing the activities of the specified system within the organization of which it is a part;

– the *information* viewpoint, which is concerned with the kinds of information handled by the system and constraints on the use and interpretation of that information;

– the *computational* viewpoint, which is concerned with the functional decomposition of the system into a set of objects that interact at interfaces – enabling system distribution;

– the *engineering* viewpoint, which is concerned with the infrastructure required to support system distribution;

– the *technology* viewpoint, which is concerned with the choice of technology to support system distribution.

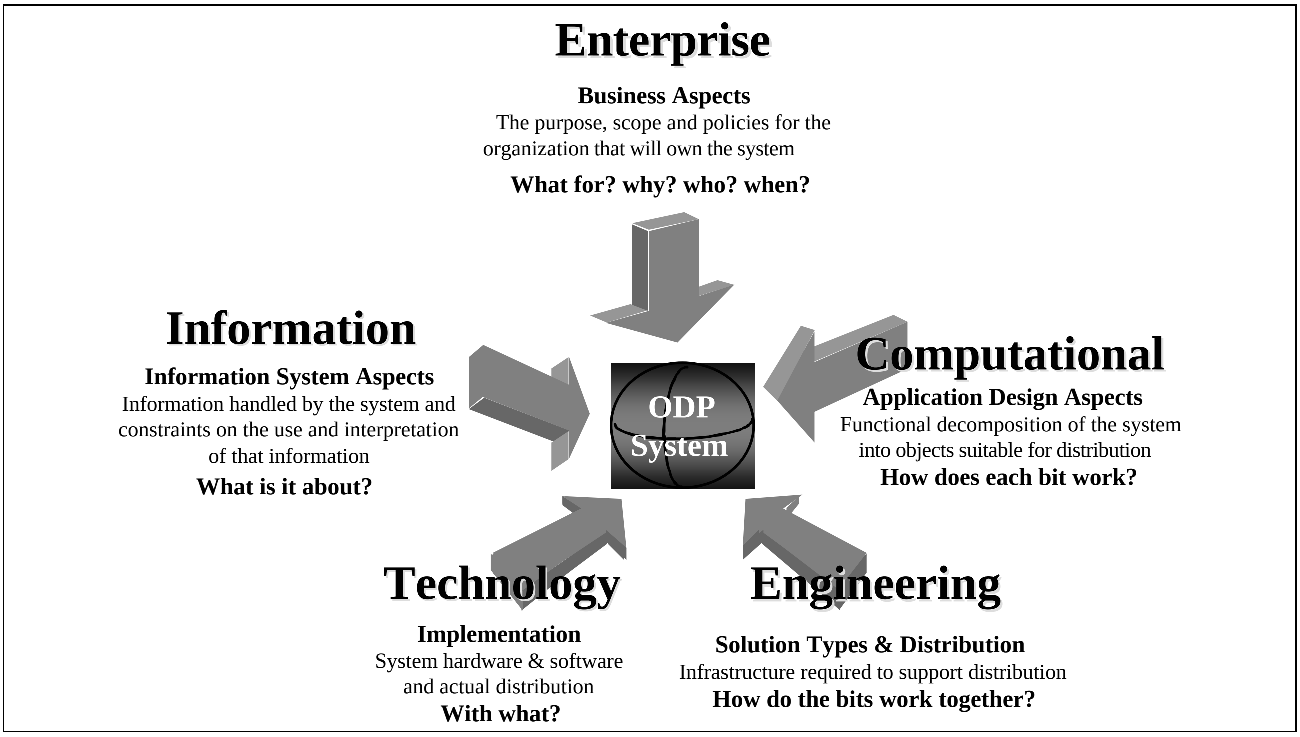



**Figure 2 – RM-ODP viewpoints**

For each *viewpoint* there is an associated *viewpoint language* which can be used to express a specification of the system from that viewpoint. The object modelling concepts give a common basis for the *viewpoint languages* and make it possible to identify relationships between the different *viewpoint* specifications and to assert correspondences between the representations of the system in different *viewpoints* (see [6.7]).

NOTE – Although the different viewpoints can be independently defined and there is no explicit order imposed by the RM-ODP for specifying them, a common practice is to start by developing the *enterprise* specification of the system, and then prepare the *information* and *computational* specifications. These two specifications may have constraints over each other. An iterative specification process is quite common too, whereby each *viewpoint* specification may be revised and refined as the other two are developed. Correspondences between the elements of these three *viewpoints* are defined during this process. After that, the *engineering* specification of the system is prepared, based on the *computational* specification. Correspondences between the elements of these *viewpoints* are then defined together with the newly specified elements. Finally, the *technology* specification is produced based on the *engineering* specification. Again, some refinements may be performed on the rest of the *viewpoint* specifications, due to the new requirements and constraints imposed by the particular selection of technology.

### 6.3.3 Distribution transparency

Distribution transparencies enable complexities associated with system distribution to be hidden from applications where they are irrelevant to their purpose. For example:

- *access transparency* masks differences of data representation and invocation mechanisms for services between systems;
- *location transparency* masks the need for an application to have information about location in order to invoke a service;
- *relocation transparency* masks the relocation of a service from applications using it;
- *replication transparency* masks the fact that multiple copies of a service may be provided in order to provide reliability and availability.

ODP standards define functions and structures to realize distribution *transparencies*. However, there are performance and cost tradeoffs associated with each *transparency* and only selected *transparencies* will be relevant in many cases. Thus, a conforming ODP system shall implement those *transparencies* that it supports in accordance with the relevant standards, but it is not required to support all *transparencies*.

### 6.3.4 Conformance

The basic characteristics of heterogeneity and evolution imply that different parts of a distributed system can be purchased separately, from different vendors. It is therefore very important that the behaviours of the different parts of a system are clearly defined, and that it is possible to assign responsibility for any failure to meet the system's specifications.

The framework defined to govern the assessment of conformance addresses these issues. RM-ODP Part 2 defines four classes of reference points: programmatic reference point, perceptual reference point, interworking reference point, and interchange reference point. The reference points in those classes are the candidate for conformance points. Part 2 covers:

- identification of the reference points within an architecture that provide candidate conformance points within a specification of testable components;
- identification of the conformance points within the set of viewpoint specifications at which observations of conformance can be made;
- definition of classes of conformance point;
- specification of the nature of conformance statements to be made in each viewpoint and the relation between them.

### 6.3.5 Enterprise language

The enterprise language provides the modelling concepts necessary to represent an *ODP system* in the context of the business or organisation in which it operates. An enterprise specification defines the purpose, *scope*, and *policies* of an *ODP system* and it provides the basis for checking conformance of system implementations. The purpose of the system is defined by the specified *behaviour* of the system while *policies* capture further restrictions of the *behaviour* between the system and its environment, or within the system itself related to the business decisions of the system owners.

NOTE – An enterprise specification of a system may therefore be thought of as a statement of the “requirements” for the system. However, it must be emphasised that it is not fundamentally different from any other element of the specification for the system.

In an enterprise specification the system is represented by one or more *enterprise objects* within the *communities* of *enterprise objects* that represent its environment, and by the *roles* in which these objects are involved. These *roles* represent, for example, the users, owners and providers of information processed by the system.

### 6.3.6 Information language

The individual components of a distributed system should share a common understanding of the information they communicate when they interact, or the system will not behave as expected. Some of these items of information are handled, in one way or another, by many of the objects in the system. To ensure that the interpretation of these items is consistent, the information language defines concepts for the specification of the meaning of information stored within, and manipulated by, an ODP system, independently of the way the information processing functions themselves are to be implemented.

Information held by the *ODP system* about entities in the real world, including the *ODP system* itself is represented in an information specification in terms of *information objects*, and their relationships and *behaviour*. Basic information elements are represented by atomic *information objects*. More complex information is represented as composite *information objects* each expressing relationships over a set of constituent *information objects*.

Just as in familiar data modelling, the information specification comprises a set of related schemata, namely, the *invariant*, *static* and *dynamic* schemata:

– An *invariant schema* expresses relationships between *information objects* which must always be true, for all valid behaviour of the system.

– A *static schema* expresses assertions which must be true at a single point in time. A common use of static schemata is to specify the initial *state* of an *information object*.

– A *dynamic schema* specifies how the information can evolve as the system operates.

### 6.3.7 Computational language

The computational viewpoint is directly concerned with the distribution of processing but not with the interaction mechanisms that enable distribution to occur. The computational specification decomposes the system into *objects* performing individual functions and interacting at well-defined *interfaces*. It thus provides the basis for decisions on how to distribute the jobs to be done, because *objects* can be located independently assuming communications mechanisms can be defined in the engineering specification to support the behaviour at the *interfaces* to those *objects*.

The heart of the computational language is the computational object model which constrains the computational specification by defining:

– the form of *interface* an *object* can have;

– the way that *interfaces* can be bound and the forms of *interaction* that can take place at them;

– the *actions* an *object* can perform, in particular the creation of new *objects* and *interfaces*, and the establishment of *bindings*.

The computational object model provides the basis for ensuring consistency between different engineering and technology specifications (including programming languages and communication mechanisms) since they must be consistent with the same computational object model. This consistency allows open interworking and portability of components in the resulting implementation.

The computational language enables the specifier to express constraints on the distribution of an application (in terms of environment contracts associated with individual *interfaces* and *interface bindings* of *computational objects*) without specifying the actual degree of distribution in the computational specification — which is specified in the engineering and technology specifications. This ensures that the computational specification of an application is not based on any unstated assumptions affecting the distribution of *engineering* and *technology objects*. Because of this, the configuration and degree of distribution of the hardware on which ODP applications are run can easily be altered, subject to the stated environment constraints, without having a major impact on the application software.

### 6.3.8 Engineering language

The engineering language focuses on the way object interaction is achieved and on the resources needed to do so. It defines concepts for describing the infrastructure required to support selective distribution transparent interactions between *objects*, and rules for structuring communication *channels* between *objects* and for structuring systems for the purposes of resource management. These rules can be expressed as *engineering templates* (for example *engineering channel template*).

Thus the computational viewpoint is concerned with when and why *objects* interact, while the engineering viewpoint is concerned with how they interact. In the engineering language, the main concern is the support of *interactions* between *computational objects*. As a consequence, there are very direct links between the viewpoint descriptions: *computational objects* are visible in the engineering viewpoint as *basic engineering objects* and computational bindings, whether implicit or explicit, are visible as either *channels* or local *bindings*.

The concepts and rules are sufficient to enable specification of internal interfaces within the infrastructure, enabling the definition of distinct conformance points for different transparencies, and the possibility of standardization of a generic infrastructure into which standardized transparency modules can be placed.

NOTE – The engineering language assumes a virtual machine that corresponds to a computing environment offering minimal support for distribution (e.g. a set of computing systems with stand-alone OS facilities plus communication facilities). In practice, the functionality available from current vendor technology, for example when it offers a CORBA or J2EE environment, already provides significant elements of the functionality to be covered by the engineering specification.

Thus, the engineering specification is interpreted in this document as defining the mechanisms and functions required to support distributed interaction between objects in an ODP system, making use of the supporting functionality provided by the specific vendor technology defined by the technology specification.

#### 6.3.9 Technology language

The technology specification describes the implementation of the ODP system in terms of a configuration of *technology objects* representing the hardware and software components of the implementation. It is constrained by cost and availability of technology objects (hardware and software products) that would satisfy this specification. These may conform to *implementable standards* which are effectively templates for *technology objects*. Thus, the technology viewpoint provides a link between the set of viewpoint specifications and the real implementation, by listing the standards used to provide the necessary basic operations in the other viewpoint specifications, and the aim of the technology specification is to provide the extra information needed for implementation and testing by selecting standard solutions for basic components and communication mechanisms. Such a selection is necessary to complete the system specification, but is largely divorced from the rest of the design process.

### 6.4 Overview of UML modelling concepts

The Unified Modelling Language (UML) is a visual language for specifying, constructing and documenting the artefacts of systems. It is a general-purpose modelling language that can be used with all major object and component methods and that can be applied to all application domains (e.g., health, finance, telecom, aerospace) and implementation platforms (e.g., J2EE, CORBA, .NET). However, not all of UML modelling capabilities are necessarily useful in all domains or applications. Therefore, UML 2.0 has been structured modularly, with the ability to select only those parts of the language that are of direct interest, and extensible, so it can be easily customized.

UML 2.0 defines twelve types of diagrams, divided in three categories that represent, respectively: the static application structure; different aspects of dynamic behaviour; and three ways for organizing and managing the application modules. In addition, UML 2.0 incorporates powerful extension mechanisms that allow the definition of new dialects of UML to customize the language for particular platforms and domains.

#### 6.4.1 Structural models

Structural models specify the structure of objects in a model. They are represented in:

- class diagrams, which show a collection of declarative (static) model elements, such as classes, types, and their contents;
- object diagrams, which encompass objects and their relationships at a point in time. An object diagram may be considered a special case of a class diagram or a communication diagram;
- component diagrams, which show the organizations and dependencies among components;
- deployment diagrams, which represent the execution architecture of systems. They represent system artifacts as nodes, which are connected through communication paths to create network systems of arbitrary complexity. Nodes are typically defined in a nested manner, and represent either hardware devices or software execution environments;
- composite structure diagrams, which depict the internal structure of a classifier, including the interaction points of the classifier to other parts of the system. They show the configuration of parts that jointly perform the behaviour of the containing classifier.
- package diagrams, which depict how model elements are organized into packages and the dependencies among them, including package imports and package extensions.

#### 6.4.2 Behavioural models

Behavioural models specify the behaviour of objects in a model. They are represented by:

- use case diagrams, each of which illustrates the relationships among actors and the system, and use cases;
- statechart diagrams, which depict discrete behaviour modelled through finite state-transition systems. In particular, a state machine diagram specifies the sequences of states that an object or an interaction goes through during its life in response to events, together with its responses and actions;

– activity diagrams, which depict behaviour using a control and data-flow model;
– interaction diagrams, which emphasize object interactions, can be one of the following:
  – collaboration diagrams, each of which represents a configuration of objects interacting for a given set of purposes;
  – sequence diagrams, that depict interactions by focusing on the sequence of messages that are exchanged, along with their corresponding event occurrences on the lifelines. Unlike a communication diagram, a sequence diagram includes time sequences but does not include object relationships. A sequence diagram can exist in a generic form (describes all possible scenarios) and in an instance form (describes one actual scenario). Sequence diagrams and communication diagrams express similar information, but show it in different ways;
  – communication diagrams, which focus on the interaction between lifelines where the architecture of the internal structure and how this corresponds with the message passing is central. The sequencing of messages is given through a sequence numberering scheme. Sequence diagrams and communication diagrams express similar information, but shown it in different ways.
  – interaction overview diagrams, which represent interactions through a variant of activity diagrams in a way that promotes overview of the control flow, and where each node can be an interaction diagram;
  – timing diagrams, which show the change in state or condition of a lifeline (representing a classifier instance or classifier role) over linear time. The most common usage is to show the change in state of an object over time in response to accepted events or stimuli.

### 6.4.3 Model management

Model management concerns the structuring of a model in terms of the groupings of model elements that comprise it. There are four grouping elements:

– Models, which are used to capture different views of a physical system;
– Packages, which are used within a model to group model elements;
– Subsystems, which represents behavioural units in the physical system being modelled;
– Profiles, which are packages grouping UML extensions.

### 6.4.4 Extension mechanisms

UML 2.0 provides a rich set of modelling concepts and notations that have been carefully designed to meet the needs of typical software modelling projects. However, users may sometimes require additional features beyond those defined in the UML standard.

UML 2.0 can be extended in two ways. First, a new dialect of UML can be defined by using Profiles to customize the language for particular platforms (e.g., J2EE/EJB, .NET/COM+) and domains (e.g., finance, telecommunications, aerospace). Alternatively, a new language related to UML can be specified by reusing part of the UML 2.0 InfrastructureLibrary package and augmenting with appropriate metaclasses and metarelationships. The former case defines a new dialect of UML, while the latter case defines a new member of the UML family of languages.

A Profile is a kind of Package that extends a reference metamodel. The primary extension construct is the Stereotype, which defines how an existing metaclass may be extended, and enables the use of platform or domain specific terminology or notation in place of or in addition to the ones used for the extended metaclass. Just like a class, a stereotype may have properties, which are referred to as tag definitions. When a stereotype is applied to a model element, the values of the properties are referred to as tagged values.

Constraints are frequently defined in a profile, and typically define well-formedness rules that are more constraining (but consistent with) those specified by the reference metamodel. The constraints that are part of the profile are evaluated when the profile has been applied to a package, and need to be satisfied in order for the model to be well formed.

NOTE – In this document, stereotypes are used to represent domain specific specializations of UML 2.0 metaclasses in order to represent the semantics of the RM-ODP viewpoint language concerned.

## 6.5 General principles for expressing and structuring ODP system specifications using UML

This clause defines the structuring style for ODP system specifications, expressed using the UML profiles defined in Clauses 7 to 11 of this document. ODP system specifications that are in compliance with this document will use this structuring style.

The ODP system specification will consist of a single UML model stereotyped as «ODP_SystemSpec», that contains a set of UML models, one for each viewpoint specification, each stereotyped as «<X>_Spec», where <X> is the viewpoint concerned.

Each viewpoint model will be expressed in the corresponding viewpoint language, using the appropriate UML profile for that language, as described in Clauses 7 to 11 of this document.

In general, using the UML to represent a given viewpoint specification (which will consist of a coherent set of instances of the concepts described in each viewpoint language) requires that:

- suitable mappings be identified from each of the viewpoint language concepts to one or more appropriate UML sub-typed metaclasses (expressed by the use of stereotypes), and that
- the relationships (meta-associations) between the viewpoint language concepts (e.g. "a *community* has exactly one *objective*", in the enterprise language) be similarly represented, preferably by meta-associations between the corresponding UML metaclasses (e.g. "Class may be associated with Class") or, failing that, by use of specific additional UML model elements.

This must be done in a way that is consistent with the semantics implicit in the UML metamodel.

In addition, a set of traces between elements from different UML models will specify the *correspondences* between the corresponding ODP modelling elements (see [6.7]).

Temporary Note – This way of representing correspondences might change as the result of the NB contributions for Clause 12.

## 6.6 Conformance in UML specifications of ODP systems

Conformance relates an implementation to a specification. Any proposition that is true in the specification must be true in its implementation. A conformance statement is a statement that identifies conformance points of a specification and the behaviour which must be satisfied at these points. Conformance statements will only occur in specifications which are intended to constrain some feature of a real implementation, so that there exists, in principle, the possibility of testing.

The RM-ODP identifies certain reference points in the architecture as potentially declarable as conformance points in specifications. That is, as points at which conformance may be tested and which will, therefore, need to be accessible for test. However, the requirement that a particular reference point be considered a conformance point must be stated explicitly in the conformance statement of the specification concerned, together with the conformance criteria that should be satisfied at this point.

Reference points will be identified in the UML specification by the use of the stereotype «ODP_ReferencePoint» (which extends UML 2.0 metaclass element) on the model elements that map to the corresponding reference points. Conformance statements will be mapped to UML comments stereotyped «ODP_ConformanceStatement», attached to the UML model elements (stereotyped «ODP_ReferencePoint») that map to the corresponding reference points. These comments will describe the conformance criteria that should be satisfied at the reference point. Therefore, conformance criteria are those model elements stereotyped «ODP_ReferencePoint», which have also attached a «ODP_ConformanceStatement» comment. It is possible to attach multiple «ODP_ConformanceStatement» comments to one model element stereotyped «ODP_ReferencePoint», thus declaring several conformance criteria at the same reference point.

## 6.7 Correspondences between viewpoint specifications

### 6.7.1 ODP Correspondences

The correspondences between viewpoint specifications are defined in Part 3 of the RM-ODP and in the Enterprise Language. The text that follows in this clause is abstracted from these standards, which remain the authoritative standards, and should be followed in case of conflicts between this document and those standards.

A set of specifications of an ODP system written in different viewpoint languages should not make mutually contradictory statements i.e., they should be mutually consistent. Thus, a complete specification of a system includes statements of correspondences between terms and language constructs relating one viewpoint specification to another viewpoint specification, showing that the consistency requirement is met.

The key to consistency is the idea of correspondences between different viewpoint specifications, i.e., a statement that some terms or structures in one specification correspond to other terms and specifications in a second specification. The underlying rationale in identifying correspondences between different viewpoint specifications of the same ODP system is that there are some entities that are represented in one viewpoint specification, which are also represented in another viewpoint specification. The requirement for consistency between viewpoint specifications is driven by, and only by, the fact that what is specified in one viewpoint specification about an entity needs to be consistent with what is said about the same entity in any other viewpoint specification. This includes the consistency of that entity's properties, structure and behaviour.

The specifications produced in different ODP viewpoints are each complete statements in their respective languages, with their own locally significant names, and so cannot be related without additional information in the form of correspondence statements that make clear how constraints from different viewpoints apply to particular elements of a single system to determine its overall behaviour. The correspondence statements are statements that relate the various different viewpoint specifications, but do not form part of any one of the five basic viewpoints. The correspondences can be established in two ways:

– by declaring correspondences between terms in two different viewpoint languages, stating how their meanings relate. This implies that the two languages are expressed in such a way that they have a common, or at least a related, set of foundation concepts and structuring rules. Such correspondences between languages necessarily imply and entail correspondences relating to all things of interest which the languages are used to model (e.g. things modelled by objects or actions);

– by considering the extension of terms in each language, and asserting that particular entities being modelled in the two specifications are in fact the same entity. This relates the specifications by identifying which observations need to be interpretable in both specifications.

The correspondence statements to be provided in a system specification are specified in Part 3 and in the Enterprise Language of the RM-ODP, and in Clauses 7 to 11 of this document. They fall into two categories:

– Some correspondences are required in all ODP specifications; these are called *required correspondences*. If the correspondence is not valid in all instances in which the concepts related occur, the specification simply is not a valid ODP specification.

– In other cases, there is a requirement that the specifier provides a list of items in two specifications that correspond, but the content of this list is the result of a design choice; these are called *required correspondence statements*.

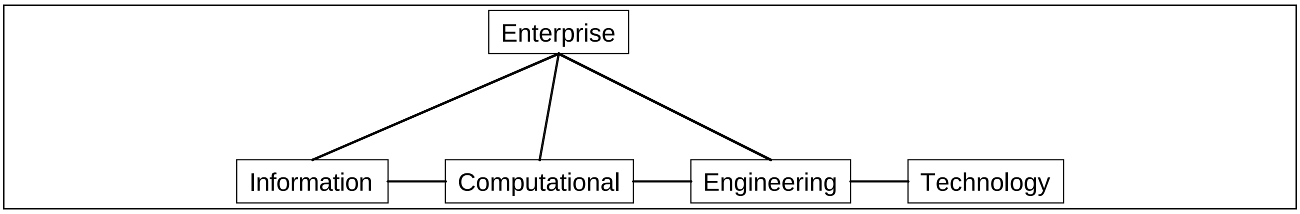



**Figure 3 – Correspondences between RM-ODP viewpoints**

NOTE – Correspondences between viewpoint specifications are illustrated in Figure 3. In RM-ODP Part 3, the following correspondences are specified.

– Between computational and information ([Part 3 – 10.1])

– Between engineering and computational ([Part 3 – 10.2])

In the Enterprise Language standard, the following correspondences are specified.

– Between enterprise and information ([E/L – 11.2])

– Between enterprise and computational ([E/L – 11.3])

– Between enterprise and engineering ([E/L – 11.4])

Note as well that the RM-ODP is silent about the correspondences between the engineering and technology viewpoint specifications.

### 6.7.2 Expressing ODP correspondences in UML

For checking the consistency between specifications, UML traces can be used to show the correspondences of UML elements of different viewpoint specifications, so that the ODP consistency rules can be checked. Thus, the application of specifications from the enterprise viewpoint to another (for example, between *enterprise objects* and *information objects*, or between enterprise *policies* and the information *schemata*) will be expressed as UML traces between the corresponding UML model elements. The interactions at these *conformance points* can then be interpreted in enterprise language terms to check that they are consistent with the *community contract* and the *policies* defined for the system.

Temporary Note – The contents of this clause may change as the result of the NB contributions requested for Clause 12.

# 7 Enterprise Specification

## 7.1 Modelling concepts

The modelling concepts used in an enterprise specification are defined, together with the structuring rules for their use, in Clause 6 of Part 3 of RM-ODP, and in the Enterprise Language. The explanations of the concepts in the text that follows are not normative, and in case of conflict between these explanations and the text in Part 3 or the Enterprise Language, the latter documents should be followed.

### 7.1.1 System concepts

An enterprise specification describes an *ODP system* (a kind of *enterprise object*) and relevant aspects of its environment. The *ODP System* has a *scope***,** which defines the *behaviour* that the system is expected to exhibit. An enterprise specification has a *field of application* which describes its usability properties.

### 7.1.2 Community concepts

A *community* is a configuration of *enterprise objects*, formed to meet a single *objective*, which is expressed in a *contract*. Any *objective* may be refined into a set of (sub-)*objectives***.**

A *contract* specifies the required *behaviour* of the *community*, and may specify *policies* that govern its structure or *behaviour* (see 7.1.3).

Each *enterprise object* models some entity (abstract or concrete thing of interest) in the Universe of Discourse. A particular kind of *enterprise object* is a *community object*, which represents, as a single object, an entity that is elsewhere in the model refined as a *community*.

The configuration of a *community* is expressed in the way *enterprise objects* interact in fulfilling *roles*, which are names standing for *behaviours* intended to meet the *objective* of the *community* concerned.

A *behaviour* is expressed as a collection of *actions* (things that happen), with constraints on when they occur. An *enterprise object* may be involved in (play *roles* in) an *action* in one or more of three ways:

- if it participates in the *action* it is an *actor* with respect to that *action*;
- if it is referenced (i.e. mentioned) in the *action*, it is an *artefact* with respect to that *action*;
- if it is essential to the (performance of) that *action***,** and requires allocation or may become unavailable, it is a *resource* with respect to that *action***.**

A *role* is an identifier for a *behaviour* of an enterprise object as can be observed from its interactions with its *environment* and the relationships between them. This implies that the *behaviour* of an *object* has to be viewed in the context of the corresponding *behaviour* of the *objects* with which it interacts.

*Communities* may be open or closed; that is they may or may not interact with their *environment*. Where a *role* that is in (i.e. is part of the configuration of) a *community* identifies *behaviour* that takes place with the participation of one or more *objects* that are not in that *community*, it is an *interface role*.

The expression of *behaviour* may be structured into one or more *processes*, each of which is a graph of *steps* taking place in a prescribed manner and leading to the fulfilment of an *objective*. In this approach, a *step* is an abstraction of an *action* in which the *enterprise objects* that participate in that *action* may be unspecified.

### 7.1.3 Policy concepts

A *policy* is a set of rules related to a particular purpose. It identifies the specification of *behaviour*, or constraints on *behaviour*, that can be changed during the lifetime of the ODP system, or that can be changed to tailor a single specification to apply to a range of different ODP systems.

The specification of a *policy* includes:

- the name of the *policy*;
- the rules, expressed as *obligations*, *permissions*, *prohibitions* and *authorizations*;
- the elements of the enterprise specification affected by the *policy*;
- *behaviour* for changing the *policy*.

The *policies* of a *community* are specified in its *contract*.

Where there is a requirement to model dynamic policy setting, a *policy* can be changed by a *behaviour*.

A *policy* may also constrain the structure (configuration) of a *community***,** by governing the assignment of *roles* to *enterprise objects*. Such a *policy* is called an *assignment policy*.

A *violation* is a *behaviour* contrary to a rule, i.e. contrary to some element in a *policy*.

A *policy* may be about Quality of Service (QoS) and other extra-functional requirements such as security, robustness, scalability, etc.

A *policy* is defined in Part 2 of RM-ODP as a "set of rules related to a particular purpose", and this concept is refined in the Enterprise Language to indicate that it is intended to be used where the desired *behaviour* of the system may be changed to meet particular circumstances (the "particular purpose" referred to in the definition). However, this fails to make any distinction between two uses of the concept. On the one hand it is appropriate to use the concept to refer to a particular set of rules, all of which apply together at some moment in time, and which implement a particular business or operational decision. On the other hand it may be used to refer to a more general set of rules that determine what *policy* values are acceptable.

Since both these concepts meet the RM-ODP definition of *policy*, it is considered better to introduce two new terms, rather than use *policy* for one and a new term for the other. Thus the first concept (the set of rules in force at some particular time) is called "policy value" and the second concept (the whole set of rules, and their relation to the particular concern involved) is called "policy envelope". In this document specific UML mappings have been developed for these two concepts and to their relations with other enterprise language concepts.

The changes to the enterprise language model that arise from this refinement of the policy concept are illustrated in Figure 8.

Temporary Note – National bodies are invited to suggest alternative approaches, for example involving the introduction of only a single new term.

### 7.1.4 Accountability concepts

Accountability concepts concern the modelled behaviour of *parties*. A *party* is an *enterprise object* modelling a natural person or any other entity considered to have some of the rights, powers and duties of a natural person, and which can therefore be considered accountable for its *actions*. A *party* may delegate authority to another *enterprise object* (which may or may not be a *party***)**, in which case it is referred to as the *principal* in that *action* of *delegation***,** and the *enterprise object* to whom authority is delegated is the *agent* of that *party*.

Only *parties* can take part in accountable *actions*. Such *actions* may take the following forms:

- *prescription*: an *action* that establishes a rule;
- *commitment*: an *action* resulting in an obligation by one or more of the participants in the act to comply with a rule or perform a *contract*;
- *declaration*: an *action* that establishes a state of affairs in the *environment* of the *object* making the *declaration*;
- *delegation*: an *action* that assigns authority, responsibility or a function to another *object*.

### 7.1.5 Structure of an enterprise specification

An enterprise specification is structured in terms of *communities* and *community objects*.

Each *community* is modelled in terms of the following concepts and the relationships between them:

- the *objective* and sub-objectives (of the *community*),
- the *behaviour* of the *community*, expressed in terms of *actions* and constraints on the order in which they may occur. Expression of *behaviour* can be structured to emphasise:
  - *roles* fulfilled by *enterprise objects* that interact as members of the *community*,
  - *processes* that represent sequences of *actions*, carried out by one or more *enterprise objects*,
- *enterprise objects* that fulfil the *roles* in the *community*,
- *policies* constraining the *behaviour*.

Some *enterprise objects* may be composite objects and are sub-classified as *community objects* and refined as *communities*.

At some level of detail the *ODP system* will be present in the model as an *enterprise object*.

### 7.1.6 Model of the enterprise language

The diagrams below illustrate the concepts of the enterprise language and the relationships between them.

NOTE – These diagrams are not identical to those in Annex A of the Enterprise Language, because they have been developed according to the conventions agreed for the UML diagrams of this document. The main difference is that these diagrams use nouns for association end names rather than verbs, and association end names are omitted when the name of the class at the end of the association is representative enough as role name for the association end. In addition, some of the n-ary associations in the former document have been replaced by semantically equivalent association classes, as this is believed to be clearer. A technical corrigendum to the Enterprise Language is being prepared.

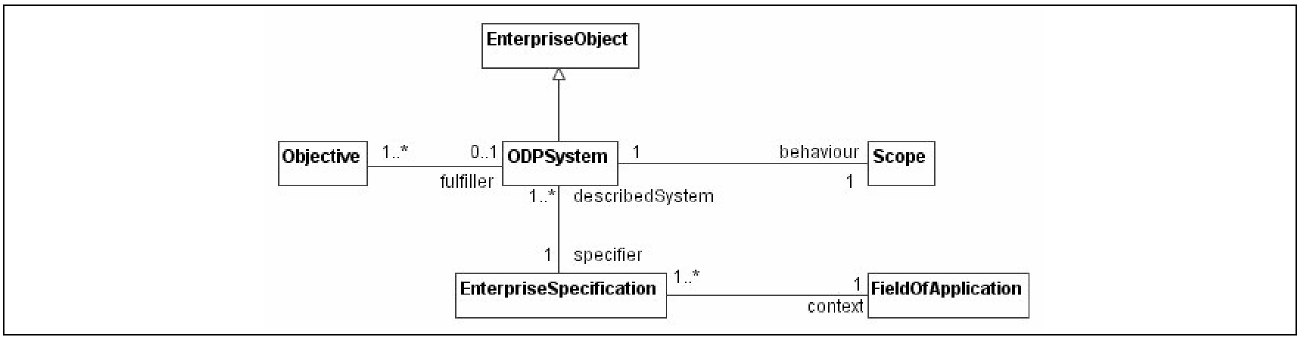



**Figure 5 – Enterprise concepts**

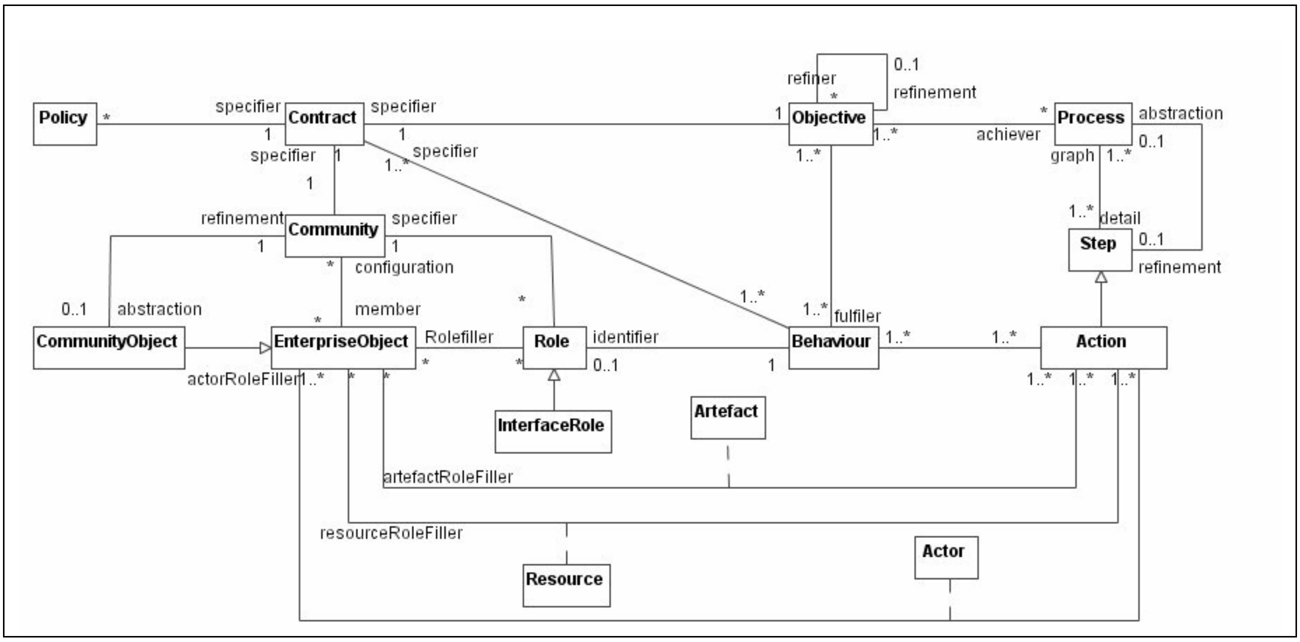



**Figure 6 – Community concepts**

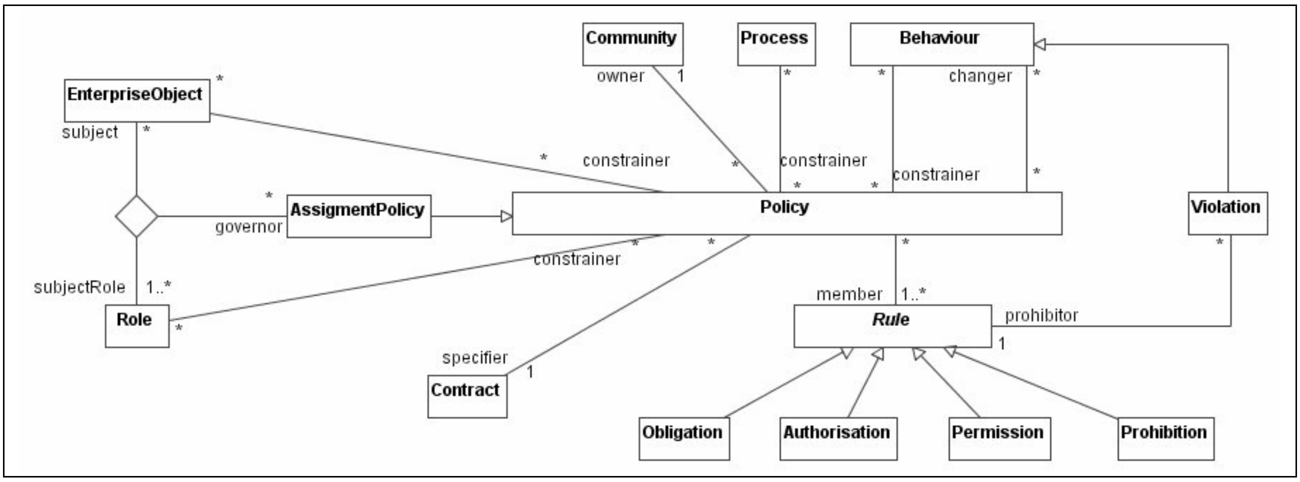



**Figure 7 – Policy concepts**

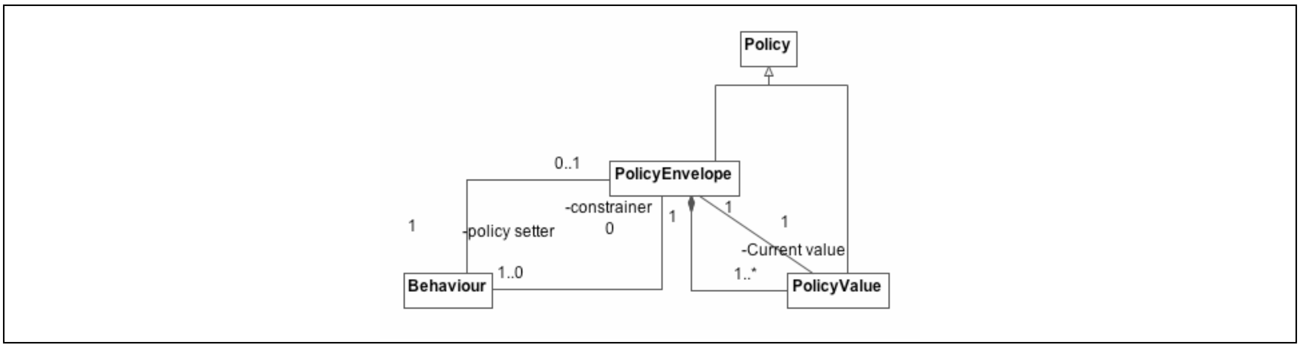



**Figure 8 – Additional policy concepts**

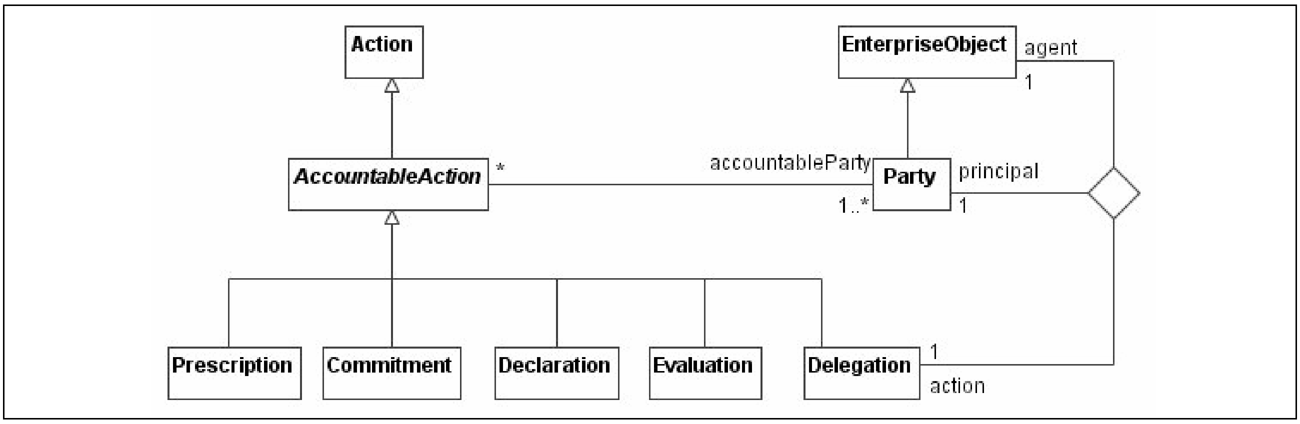



**Figure 9 – Accountability concepts**

## 7.2 UML mappings

NOTE – In this clause mappings are only defined for those concepts for which use has been demonstrated through an example, included in the main body of this document or in its annexes. Where no example has been identified, the concept concern is mentioned, but no mapping is offered.

The following paragraphs describe how the ODP enterprise concepts described in the previous Clause are represented in UML in an enterprise specification. A brief explanation of the UML concepts used in the representation of each concept is given, together with a justification of the representation used.

### 7.2.1 ODP system

An *ODP System* is a special kind of *enterprise object*. It maps to UML with a class stereotyped as «EV_ODPSystem», see [7.2.5]. Note also that modelling purposes may require that an *ODP system* be further detailed as a *community*, in which case the *enterprise object* that represents it is classified as a *community object* and refined as a *community*, see [7.2.4].

### 7.2.2 Scope

The *scope* of an *ODP system* is the set of *behaviours* that the system is expected to exhibit, e.g. its *roles*. It does not, therefore, map to a single UML model element, but to the set of elements that represent its *behaviour*.

### 7.2.3 Field of application

The *field of application* is a property of the enterprise specification as a whole, and maps to a comment stereotyped as «EV_FieldOfApplication», attached to the UML model stereotyped as «Enterprise_Spec» which contains the enterprise specification.

### 7.2.4 Community

A *community* maps to a component stereotyped as «EV_Community», which is included in a package stereotyped as «EV_CommunityContract» that contains the specification of the *community*, i.e., its *objective*, its *behaviour*, and any *enterprise objects* that are specific to the community concerned (see [7.2.8]).

Any component mapping to a *community* will have exactly one association, stereotyped as «EV_ObjectiveOf» to a class stereotyped as «EV_Objective», that maps to the *objective* of the *community*, and a set of realizations, each stereotyped as «EV_CommunityBehaviour», to the UML classifier model elements mapping to its *roles* and the associated *behaviour* (*interactions*, *actions*, *steps* and *processes*).

See also [7.2.7] and [7.2.8].

### 7.2.5 Enterprise object

Each *enterprise object* is mapped to a class stereotyped as «EV_EnterpriseObject». Where a specific individual entity is being referenced (e.g. the *ODP system*), the class concerned is a singleton. Any class stereotyped as «EV_EnterpriseObject» may have any number of associations, each stereotyped as «EV_FulfilsRole», with any number of classes stereotyped as «EV_Role» in one or more *community*, expressing the fact that the *enterprise object* fulfils the *roles*.

NOTE – In theory, an *enterprise object* could be mapped to a UML object, but since most behavioural aspects in UML are best modelled with classifiers, this approach is not adopted in this document.

#### 7.2.6 Community object

A *community object* is an *enterprise object* that is refined in the model as a *community*. It is mapped to a class stereotyped as «EV_CommunityObject», with a dependency, stereotyped as «EV_RefinesAsCommunity», to the component stereotyped as «EV_Community» which maps to the *community* that refines it.

#### 7.2.7 Objective

An *objective* (of a *community*) is mapped to a class, stereotyped as «EV_Objective». This class has an association, stereotyped as «EV_ObjectiveOf» with the component, stereotyped as «EV_Community» that maps to the *community* being specified.

#### 7.2.8 Contract

A *contract* for a *community* specifies the *objective* of that *community*, and how that *objective* can be met (i.e., its *behaviour* and *policies*). It is the specification of that *community* as it appears in the enterprise specification. The mapping, for *contract* is to a package stereotyped as «EV_CommunityContract».

In the name space of the package will be the UML model elements mapping to the *community* itself, its *objective*, its *roles* and the associated *behaviour* (*actions*, *interactions*, *steps* and *processes*), and the policy and accountability concepts specific to the *community*. Relationships between all these UML model elements may also be included in this package's namespace. The package may also contain some or all of the elements mapping to the *enterprise objects* that fulfil its *roles*. (Those elements mapping to *enterprise objects* that fulfil *roles* in other *communities* may be contained in any one of the packages mapping to those *communities*.)

#### 7.2.9 Behaviour

##### 7.2.9.1 General

NOTE – In this clause phrases such as "*interactions* between *roles*" and "*steps* performed by *roles*" should be read as "*interactions* between *enterprise objects* fulfilling *roles*" and "*steps* performed by *enterprise objects* fulfilling *roles*" respectively.

A *behaviour* is a set of *actions* with constraints on when they may occur, and is not mapped to any single UML model element. It is expressed as a set of model elements representing the *behaviour* in terms of *interactions* between *roles* in a *community*, and/or a set of model elements representing the *behaviour* as a set of *processes* of a *community* in which the *steps* are performed by *roles* in the *community*.

##### 7.2.9.2 Behaviour as interactions between roles

Where the *behaviour* is expressed in terms of *interactions* between *roles* in a *community*, a *role* is mapped to a class stereotyped as «EV_Role», in the name space of the package (stereotyped as «EV_CommunityContract») that specifies the *community* in which the *role* is specified. The *behaviour* identified by the *role* is mapped to the following combination of UML model elements:

– One or more classes each having one or more associations with the class stereotyped as «EV_Role» mapping to the role being specified. Each of these classes is stereotyped as «EV_Interaction». These associations are stereotyped as «EV_InteractionInitiator» or «EV_InteractionResponder» depending upon the part played by the corresponding role in the interaction.

– Each class stereotyped as «EV_Interaction» will also, in general, have associations (also stereotyped as above) with other classes that are stereotyped as «EV_Role», where there is an interaction between the enterprise objects fulfilling these roles.

– An *interaction* may be defined as a composition of *interactions*. When it is not defined as a composition it has an association with a signal stereotyped as «EV_Artefact» mapping to an *artefact role* (see [7.2.11]), which also has an association with an «EV_EnterpriseObject» class, defining the information that is exchanged in the *interaction*.

– A StateMachine for which the context is the «EV_Role», that defines the constraints on the receiving and sending of information by an *enterprise object* fulfilling the *role* and any associated *internal actions* of the *enterprise object*. This StateMachine shows the sending and receiving of the signals, each stereotyped as «EV_Artefact», associated with the *interactions* of the *role,* and thus shows the logical ordering of these *interactions* and may define the *internal actions* of the *role* in terms of the behaviours associated with the states.

Annex B illustrates the application of these concepts.

#### 7.2.9.3 Behaviour as processes and steps

Where the *behaviour* is expressed in terms of *processes* of a *community*, a *process* is mapped to an activity stereotyped as «EV_Process» with a realization link, stereotyped as «EV_CommunityBehaviour» from the component, stereotyped as «EV_Community», mapping to the *community* that uses this *process* to achieve its *objective*. Within this activity:

- the *steps* of the *process* are mapped to Actions, stereotyped as «EV_Step»;
- the *roles* of the *enterprise objects* that perform the *steps* (as *actors*) are mapped to ActivityPartitions stereotyped as «EV_Role»;
- the *enterprise objects* that are referenced in the *steps* (as *artefacts*) are mapped to ObjectNodes, stereotyped as «EV_Artefact».

If there is a corresponding interaction model, the Actions in a ActivityPartition mapping to a *role* must correspond to the *internal actions* identified in (the states of) the StateMachines for the class mapping to the *role* concerned in the interaction model.

#### 7.2.9.4 Interface role

An *interface role* is mapped to a class stereotyped as an «EV_Role». The part of the *behaviour* identified by the *interface role* that takes place with the participation of one or more external *objects* (*objects* that do not form part of the decomposition of the *community object* that is refined by that *community*) is represented by an *interaction* with a *role* that identifies the required *behaviour* of the external *objects*. This *behaviour* maps to a class stereotyped as an «EV_Interaction» that has associations with each of the classes (stereotyped as «EV_Role») that map to the *interface role* on the one hand and the *role* that identifies the *behaviour* of the external *objects* on the other.

### 7.2.10 Actor (with respect to an action)

The concept *actor* is a relationship between an *enterprise object* and an *action.* There is no single UML model element that maps to an instance of the RM-ODP enterprise language concept, *actor*. *Actors* in a model may be identified from either or both of:

- an examination of the *interaction* model where the existence of *actors* will be indicated by the associations, stereotyped as «EV_FulfilsRole», between the classes stereotyped as «EV_Role» and «EV_EnterpriseObject», respectively, taken in combination with the StateMachine that represents the *behaviour* of the relevant *role*.
- in an examination of the *process* model, the presence of an «EV_Step» in an «EV_Role» ActivityPartition indicates that the *enterprise object* fulfilling the *role* is an *actor* for the *step* concerned.

### 7.2.11 Artefact (with respect to an action)

The concept *artefact* is also a relationship between an *enterprise object* and an *action.* In an *interaction* model, an *artefact* referenced in an *action* maps to a Signal stereotyped as «EV_Artefact», which has two associations:

- one association, stereotyped as «EV_ArtefactRole», will be with the «EV_EnterpriseObject» class mapping to the enterprise object that is an *artefact* with respect to the *action*;
- the other association, stereotyped as «EV_ArtefactReference», will be with the «EV_Interaction» Class that maps to the *(inter-)action* for which the *enterprise object* is an *artefact*.

In a *process* model, it is possible to map each instance of *artefact* to a single UML model element, namely an ObjectFlow stereotyped as «EV_Artefact».

### 7.2.12 Resource (with respect to an action)

No specific mappings are defined. It may be useful in a model to identify, in a comment, that some *behaviour* requires the existence of an *enterprise object* as a *resource*.

### 7.2.13 Policy

NOTE – In this clause a distinction is made between *policy* value (the particular set of rules related to some purpose that are applicable at some moment in time as a result of a business decision to apply them), and *policy* envelope (a general set of rules related to a purpose, which constrain any particular set of rules that may be applicable at any particular time). See [7.1.3].

*Policies* are expressed in UML using a combination of model elements, which together are used to express the *policy* itself, and the other model elements that represent the *objects* and the *behaviour* constrained by the *policy*, as well as the *objects* and their *behaviour* by which the *policy* value may be changed.

The *policy* envelope maps to a class stereotyped as «EV_PolicyEnvelope», with a note stereotyped as «description» which explains the policy and its rules in natural language.

Each *policy* value maps to a class stereotyped as «EV_PolicyValue» each with two associations (one an aggregation, the other a regular association in which the *policy* value has the role "current value") with the «EV_PolicyEnvelope» class that maps to the *policy* envelope that provides the context for the *policy* value.

Where the enterprise specification includes elements representing the *behaviour* concerned with setting the *policy value*, this is represented as normal by *processes* or *interactions*, with associations, stereotyped as «EV_ControllingAuthority», between the classes mapping to the *policy envelope* and the classes mapping to the *behaviour*.

The relationships between a policy envelope and the behaviour it constrains are expressed by one or more dependencies, stereotyped as «EVAffectedBehaviour», from the classes mapping to the *behaviour* to the class mapping to the *policy envelope*.

Unless the set of *policy values* is pre-determined, a set of constraints stereotyped as «EV_PolicyEnvelopeRule» expressing rules governing acceptable *policy* values is attached to the «EV_PolicyEnvelope» class.

Attached to each «EV_PolicyValue» class are a set of constraints stereotyped as «EV_PolicyValueRule», which together express behaviour rules related to the *policy* value. These rules may be expressed in OCL or other suitable notation.

The pattern for expression of *policy* and its impact on other parts of an enterprise specification is shown in Figure 10.

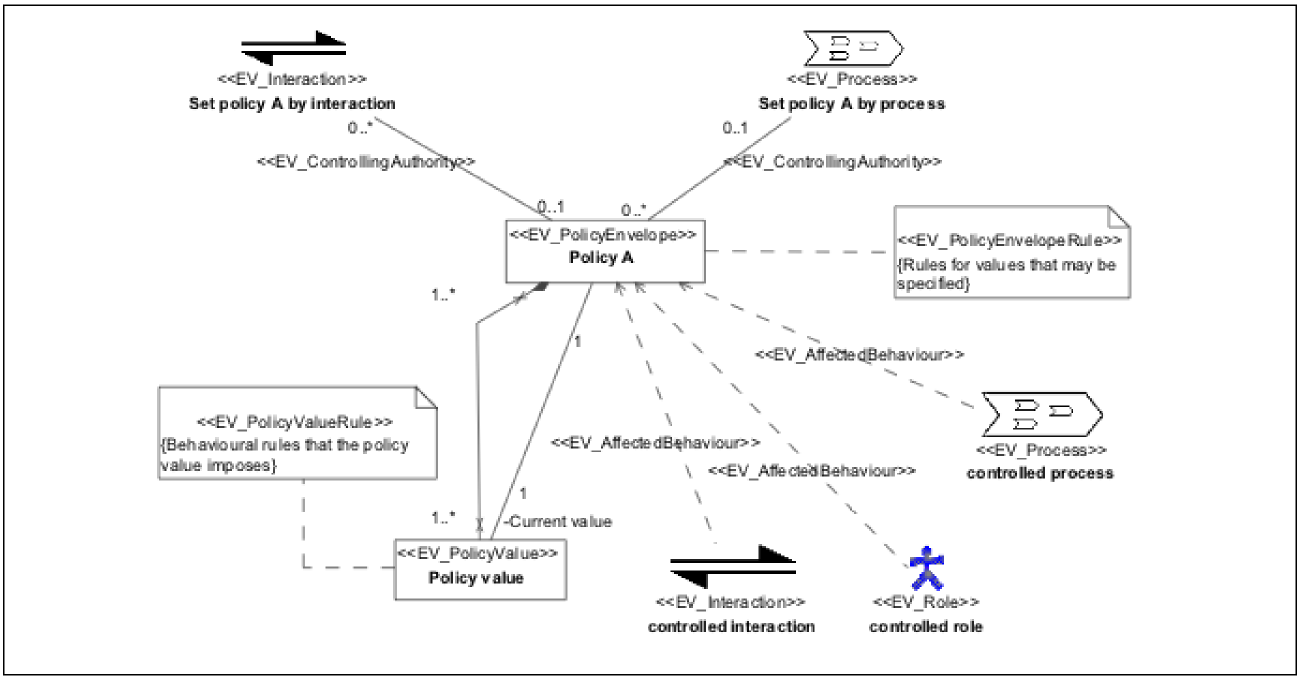



**Figure 10 – Pattern for UML expression of a *policy***

### 7.2.14 Obligation

No specific mappings are defined. It may be useful in a model to identify, in a comment, that some *behaviour* places or fulfils an *obligation*.

### 7.2.15 Authorisation

No specific mappings are defined. It may be useful in a model to identify, in a comment, that some *behaviour* requires or creates an *authorisation*.

#### 7.2.16 Permission

No specific mappings are defined. It may be useful in a model to identify, in a comment, that some *behaviour* requires or creates a *permission*.

#### 7.2.17 Prohibition

No specific mappings are defined. It may be useful in a model to identify, in a comment, that some *behaviour* requires or creates a *prohibition*.

#### 7.2.18 Assignment policy

No specific mappings are defined. An *assignment policy* is expressed in the same way as any other *policy*; see 7.2.13.

#### 7.2.19 Violation

No specific mappings are defined. It is difficult to envisage the circumstances in which a *behaviour* might be specified which is a *violation*.

#### 7.2.20 Party

A *party* is an *enterprise object* modelling an entity with some of the rights, powers and duties of a natural person. It is expressed in UML as a class stereotyped as «EV_Party», which must also be stereotyped as «EV_EnterpriseObject».

#### 7.2.21 Accountable action

An *action* may be *accountable* when it is part of the *behaviour* identified by a *role* fulfilled by a *party*. This is expressed in UML with an association, stereotyped as «EV_Accountable», between the class expressing the *role* and the class or activity expressing the *interaction* or *process* respectively in which the accountable *party* participates.

Note: Where this construyct is used for a *process*, this only indicates that the *role* is accountable for those *steps* which it performs, and not for those that performed by some other *role*. This is a limitation of the semantics of the UML approach chosen, as it is not possible to associate a classifier with the model element expressing *steps*.

#### 7.2.22 Delegation

A *Delegation* is expressed in UML by an association, stereotyped as «EV_Delegation», between two classes stereotyped as «EV_Role» with association ends showing the *party* which is the *principal* and the *enterprise object* which is the *agent* to whom the delegation is made.

#### 7.2.23 Principal

No specific mappings are defined. See [7.2.22].

#### 7.2.24 Agent

No specific mappings are defined. See [7.2.22].

#### 7.2.25 Prescription

No specific mappings are defined. It may be useful in a model to identify, in a comment, that some *behaviour* represents a *prescription*.

#### 7.2.26 Commitment

No specific mappings are defined. It may be useful in a model to identify, in a comment, that some *behaviour* represents a *commitment*.

#### 7.2.27 Declaration

No specific mappings are defined. It may be useful in a model to identify, in a comment, that some *behaviour* represents a *declaration*.

#### 7.2.28 Summary of UML Mappings for the enterprise language

The Enterprise language profile (EV_Profile) specifies how the enterprise viewpoint modelling concepts relate to and are represented in standard UML using stereotypes, tagged values, and constraints. It represents the concepts of the enterprise language model (see [7.1.6]).

The following Figures show a graphical representation of the UML Profile for the Enterprise Language, using the notation provided by UML 2.0.

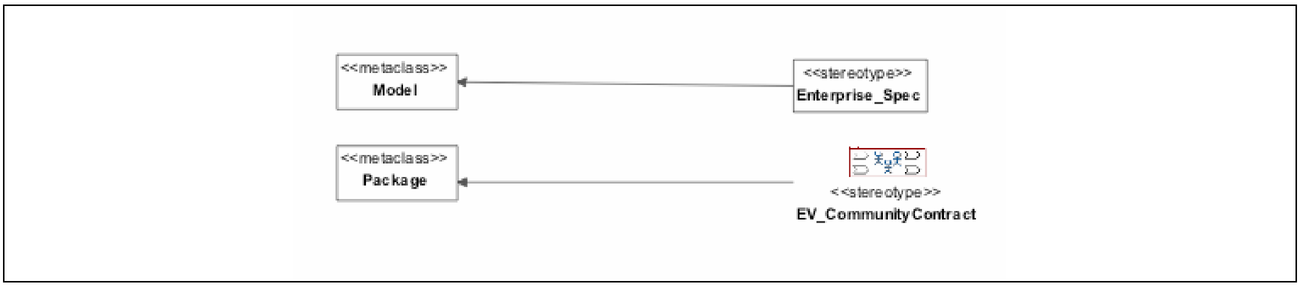



**Figure 11 – Model management**

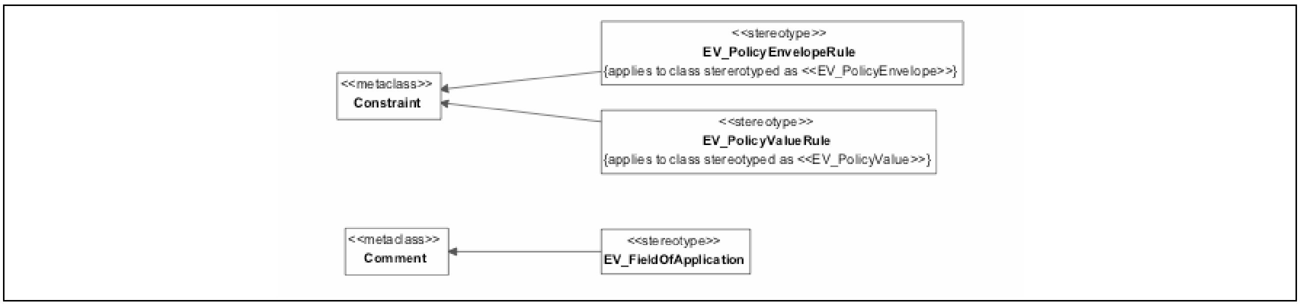



**Figure 12 – Comments and Constraints**

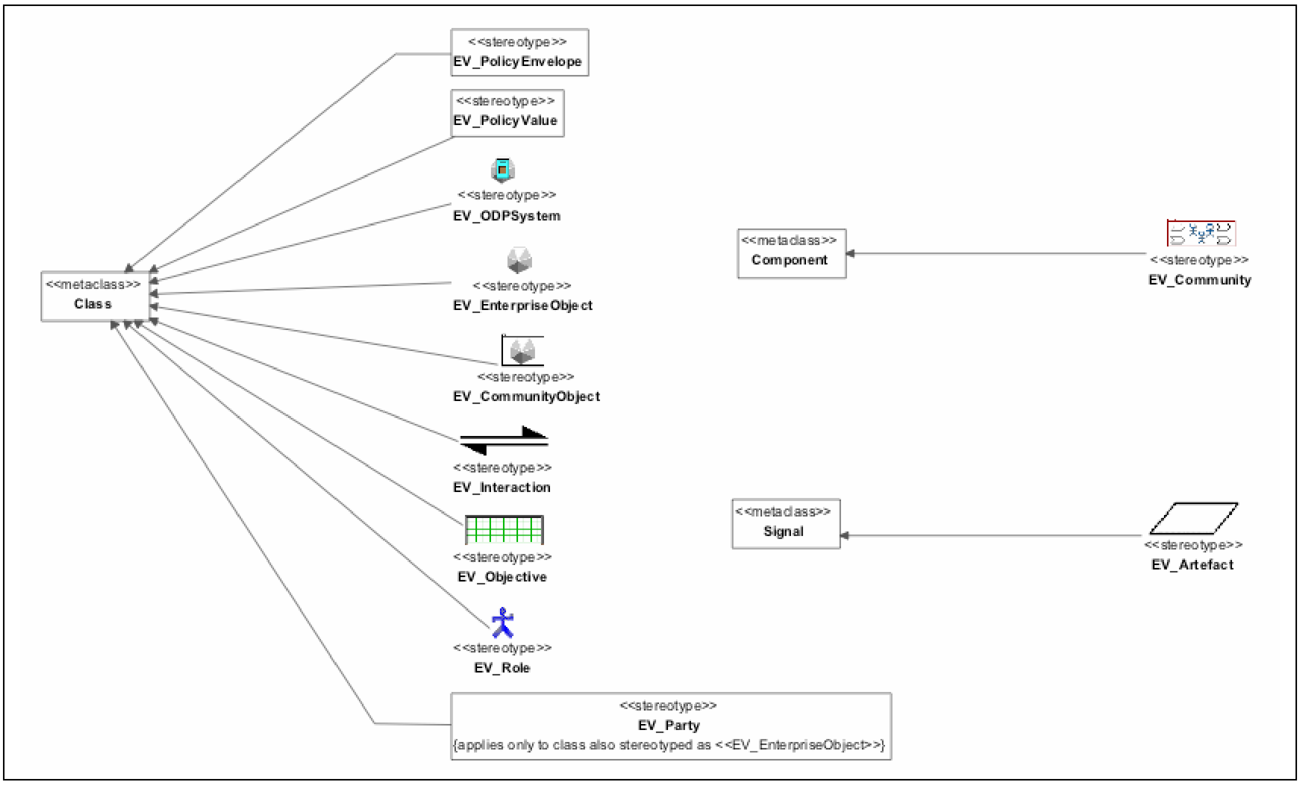



**Figure 13 – Classifiers**

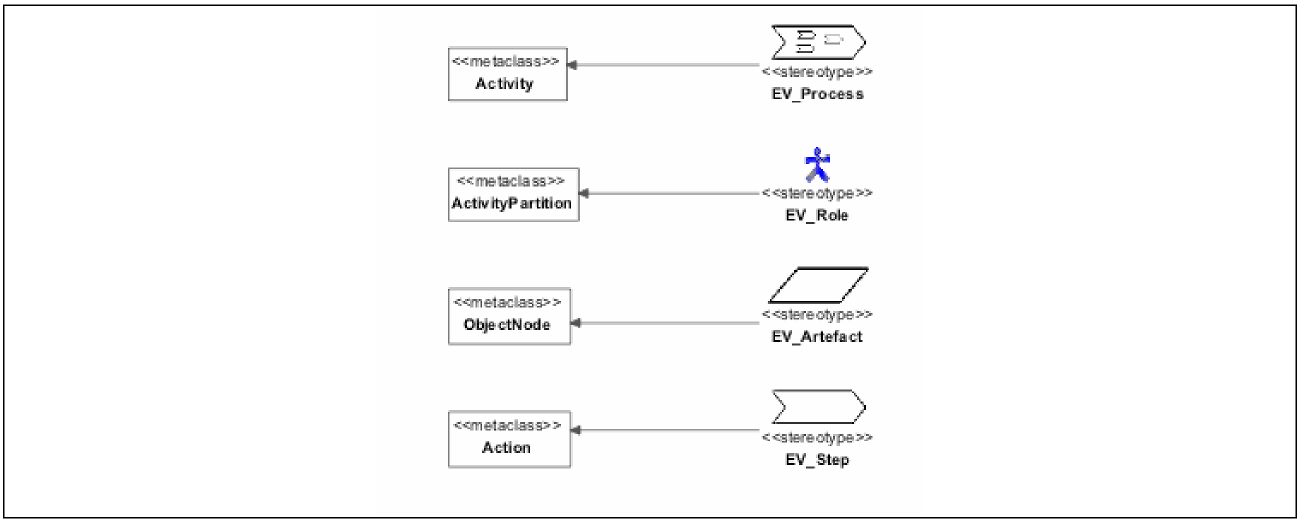



**Figure 14 – Activities**

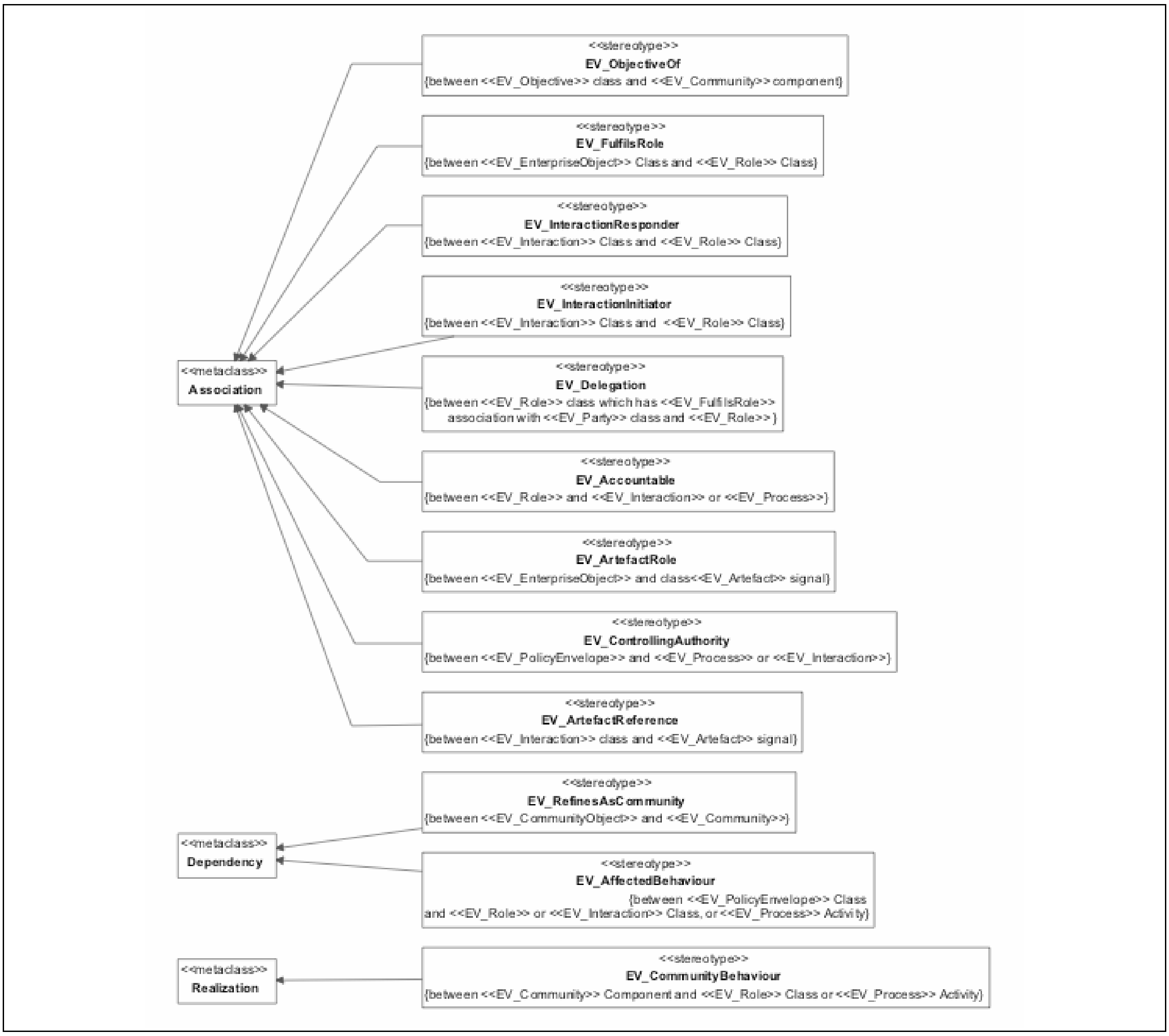



**Figure 15 – Relationships**

## 7.3 Enterprise specification structure (in UML terms)

An enterprise specification is contained in a model, stereotyped as «EnterpriseSpec». At the top level within this model there will be one or more packages, stereotyped as «EV_CommunityContract», with where necessary, associated classes, stereotyped as «EV_CommunityObject», expressing the relevant *communities* as *enterprise objects*.

Within each «EV_CommunityContract» package, there will be a single component, stereotyped as «EV_Community» and a single class, stereotyped as «EV_Objective», as well as other elements, packaged as convenient, expressing *behaviour* (*roles*, *processes* and *interactions*), and *enterprise objects* that are local to the *community*.

## 7.4 Viewpoint correspondences for the enterprise language

### 7.4.1 Enterprise and information viewpoint specification correspondences

In general, not all the elements of the enterprise specification of a system need correspond to elements of its information specification. However, the information viewpoint shall conform to the *policies* of the enterprise viewpoint and, likewise, all enterprise *policies* shall be consistent with the *static*, *dynamic*, and *invariant* schemata of the information specification.

Where there is a correspondence between enterprise and information elements (e.g., between an *enterprise object* and the *information object* that stores the relevant information about it), the specifier shall provide:

- for each *enterprise object* in the enterprise specification, a list of those *information objects* (if any) that represent information or information processing concerning the entity represented by that *enterprise object*;
- for each *role* in each *community* in the enterprise specification, a list of those *information object types* (if any) that that specify information or information processing of an *enterprise object* fulfilling that *role*;
- for each *policy* in the enterprise specification, a list of the *invariant*, *static* and *dynamic schemata* of *information objects* (if any) that correspond to the *enterprise objects* to which that *policy* applies; an *information object* is included if it corresponds to the *enterprise community* that is subject to that *policy*;
- for each *action* in the enterprise specification, the *information objects* (if any) subject to a *dynamic schema* constraining that *action*;
- for each relationship between *enterprise objects*, the *invariant schema* (if any) which constrains *objects* in that relationship;
- for each relationship between *enterprise roles*, the *invariant schema* (if any) which constrains objects fulfilling *roles* in that relationship.

### 7.4.2 Enterprise and computational viewpoint specification correspondences

Not all the elements of the enterprise specification of a system need correspond to elements of its computational specification. In particular, not all *states*, *behaviours* and *policies* of an enterprise specification need correspond to *states* and *behaviours* of a computational specification. There may exist transitional computational states within pieces of computational behaviour which are abstracted as atomic transitions in the enterprise specification.

Where there is a correspondence between enterprise and computational elements, the specifier shall provide:

- for each *enterprise object* in the enterprise specification, that configuration of *computational objects* (if any) that realizes the required *behaviour*;
- for each *interaction* in the enterprise specification, a list of those *computational interfaces* and *operations* or *streams* (if any) that correspond to the enterprise *interaction*, together with a statement of whether this correspondence applies to all occurrences of the *interaction*, or is qualified by a *predicate*;
- for each *role* affected by a *policy* in the enterprise specification, a list of the *computational object types* (if any) that exhibit choices in the computational *behaviour* that are modified by the *policy*;
- for each *interaction* between *roles* in the enterprise specification, a list of *computational binding object types* (if any) that are constrained by the enterprise *interaction*;

– for each enterprise *interaction type*, a list of computational *behaviour types* (if any) of computational *behaviours* capable of carrying out an *interaction* of that enterprise *interaction type*.

#### 7.4.3 Enterprise and engineering viewpoint specification correspondences

Not all the elements of the enterprise specification of a system need correspond to elements of its engineering specification. Where there is a correspondence between enterprise and engineering elements, the specifier shall provide:

– for each *enterprise object* in the enterprise specification, the set of those *engineering nodes* (if any) with their *nuclei*, *capsules*, and *clusters*, all of which support some or all of its *behaviour*;

– for each *interaction* between *roles* in the enterprise specification, a list of *engineering channel types* and *stubs*, *binders*, *protocol objects* and *interceptors* (if any) that are constrained by the enterprise *interaction*.

NOTE 1 – The engineering nodes may result from rules about assigning support for the behaviour of enterprise objects to nodes. These rules may capture policies from the enterprise specification.

NOTE 2 – The engineering channel types and stubs, binders or protocol objects may be constrained by enterprise policies.

#### 7.4.4 Enterprise and technology viewpoint specification correspondences

In accordance with [Part 2 – 15.5] and [Part 3 – 5.3], an implementer provides, as part of the claim of conformance, the chain of interpretations that permits observation at conformance points to be interpreted in terms of enterprise concepts. While there may be specific correspondences between enterprise *policies* and technology viewpoint specifications that require the use of particular technologies, there are neither required correspondences nor required correspondence statements.

NOTE – Although there are no required viewpoint correspondences between enterprise viewpoint and technology viewpoint specifications, there may be cases where part of enterprise viewpoint specification has a direct relationship with a technology viewpoint specification or a choice of technology. Such examples include enterprise *policies* covering performance (e.g. response time), reliability, and security.

# 8 Information Specification

## 8.1 Modelling concepts

The modelling concepts used in an information specification are defined, together with the structuring rules for their use, in Clause 6 of Part 3 of RM-ODP. The explanations of the concepts in the text that follows are not normative, and in case of conflicts between these explanations and the text in Part 3, the latter should be followed.

The information viewpoint is concerned with information modelling. It focuses on the semantics of information and information processing in the *ODP system*. The individual components of a distributed system must share a common understanding of the information they communicate when they interact, or the system will not behave as expected. Some of these items of information are handled, in one way or another, by many of the *objects* in the system. To ensure that the interpretation of these items is consistent, the information language defines concepts for the specification of the meaning of information stored within, and manipulated by, an *ODP system*, independently of the way the information processing functions themselves are to be implemented.

In general, the information language helps answer the questions “what kind of information is managed by the system?” and “what constraints and criteria need to be applied to access the information?”

In the ODP Reference Model, the information language uses a basic set of concepts and structuring rules, including those from Part 2 of RM-ODP, and three concepts specific to the information viewpoint: *invariant schema*, *static schema*, and *dynamic schema*.

### 8.1.1 Information object

Information held by the ODP system about entities in the real world, including the ODP system itself is represented in an information specification in terms of *information objects*, and their relationships and behaviour.

Basic information elements are represented by atomic *information objects*. More complex information is represented as composite *information objects* expressing relationships over a set of constituent *information objects*. *Information objects*, as any other ODP object, exhibit behaviour, state, identity, and encapsulation.

NOTE – *Information objects* may have *operations*, although information operations are names for significant stimuli for state changes, and are not necessarily the same as *computational operations*.

### 8.1.2 Information object type

The *type* of an *information object* is a predicate characterizing a collection of *information objects*.

### 8.1.3 Information object class

A *class* of *information objects* is the set of all *information objects* satisfying a given *type*.

### 8.1.4 Information object template

An *information object template* is the specification of the common features of a collection of *information objects* in sufficient detail that an *information object* can be instantiated using it. In ODP, *information object templates* may reference *static*, *invariant* and *dynamic schemata*.

### 8.1.5 Information action and action types

An *action* is a model of something that happens in the real world. In ODP, *actions* are instances, not types. Types of actions are represented by ODP *action types*. An *action* in the information viewpoint is associated with at least one *information object*.

Actions can be either *internal actions* or *interactions*. An *internal action* always takes place without the participation of the *environment* of the *object*. An *interaction* takes place with the participation of the *environment* of the *object*. *Objects* can only interact at *interfaces*. ODP *interactions* are instances of ODP *communications*.

### 8.1.6 Invariant schema

An *invariant schema* is a set of predicates on one or more *information objects* which must always be true. The predicates constrain the possible states and state changes of the *objects* to which they apply.

ODP also notes that an *invariant schema* can describe the specification of the *types* of one or more *information objects*, that will always be satisfied by whatever behaviour the *objects* might exhibit.

### 8.1.7 Static schema

A *static schema* is a specification of the state of one or more *information objects*, at some point in time, subject to the constraints of any *invariant schemata*.

A *static schema* is expressed as the specification of the *types* of one or more *information objects*, at some particular point in time, where these *types* are *subtypes* of the *types* specified in the *invariant schema*.

### 8.1.8 Dynamic schema

A *dynamic schema* is a specification of the allowable state changes of one or more *information objects*, subject to the constraints of any *invariant schemata*. A *dynamic schema* specifies how the information can evolve as the system operates. In addition to describing state changes, *dynamic schemata* can also create and delete *information objects*, and allow reclassifications of instances from one *type* to another. Besides, in the information language, a state change involving a set of *objects* can be regarded as an *interaction* between those *objects*. Not all the *objects* involved in the *interaction* need to change state; some of the *objects* may be involved in a read-only manner.

### 8.1.9 Structure of an information specification

In ODP, an information specification defines the semantics of information and the semantics of information processing in an ODP system in terms of a configuration of *information objects*, the *behaviour* of these *objects*, and *environment contracts* for the *objects* in the system. More precisely, an information specification is expressed in terms of:

- a configuration of *information objects*, described by a set of *static schemata*;
- the *behaviour* of those *information objects*, described by a set of *dynamic schemata*; and
- the constraints that apply to either of the above (*invariant schemata*).

The different schemata may apply to the whole system, or they may apply to particular domains within it. Particularly in large and rapidly evolving systems, the reconciliation and federation of separate information domains will be one of the major tasks to be undertaken in order to manage information.

There are also some considerations that need to be taken into account when specifying the information viewpoint of an ODP system:

- *Information objects* are either atomic or are represented as a composition of component *information objects*. When an *information object* is a composite *object*, the schemata are composed as well.
- Allowable state changes specified by a *dynamic schema* can include the creation of new *information objects* and the deletion of *information objects* involved in the *dynamic schema*. Allowable state changes can be subject to ordering and temporal constraints.
- The configuration of *information objects* is independent from distribution, i.e., there is no sense or focus on distribution in this viewpoint.

### 8.1.10 Model of the information language

The diagram below illustrates the concepts of the information language and the relationships between them.

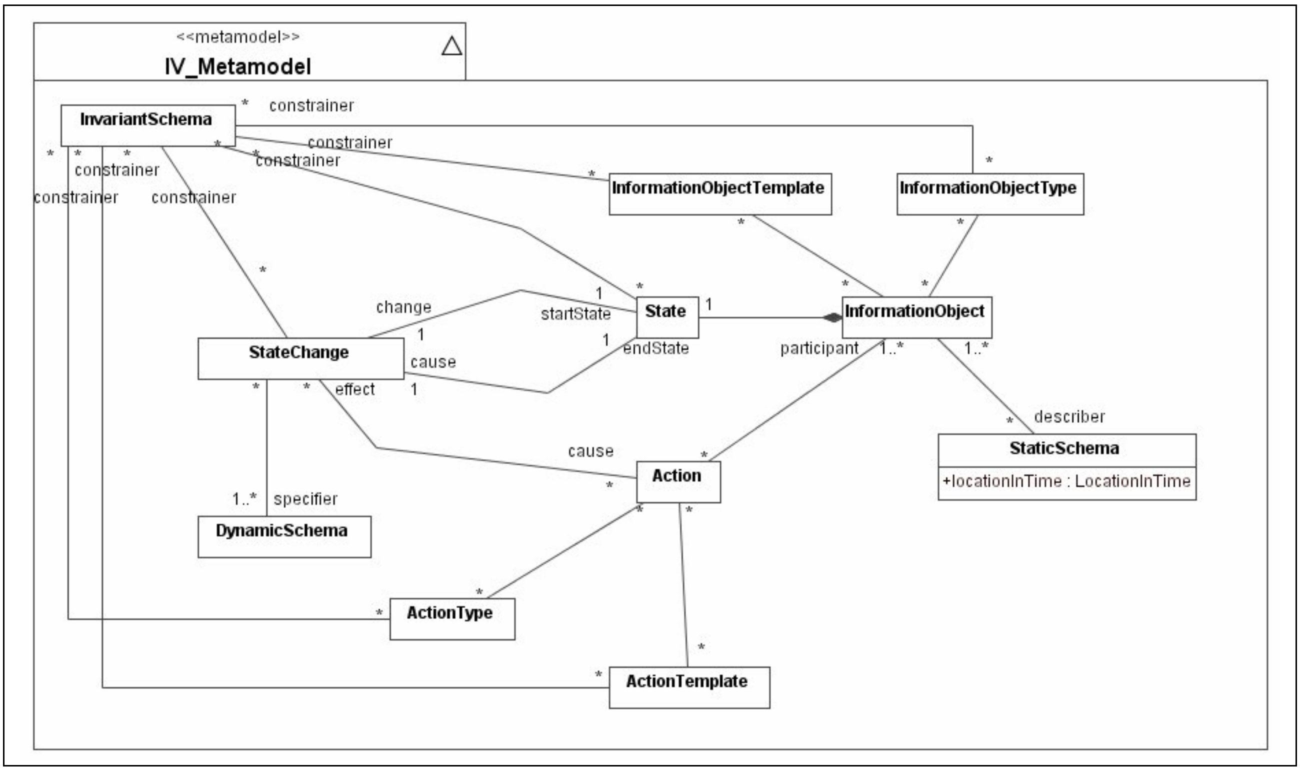



**Figure 16 – Information language concepts**

## 8.2 UML mappings

The following paragraphs describe how the ODP information concepts described in the previous Clause are represented in UML in an information specification. A brief explanation of the UML concepts used in the representation of each concept is given, together with a justification of the representation used.

NOTE – In this clause mappings are only defined for those concepts for which use has been demonstrated through an example, included in the main body of this document or in its annexes. Where no example has been identified, the concept concern is mentioned, but no mapping is offered.

### 8.2.1 Information object

An *information object* is modelled as a UML object, which is an instance of a class, and therefore it is mapped to an InstanceSpecification. An InstanceSpecification is a UML model element that represents an instance in a modelled system. It specifies existence of an entity in a modelled system and completely or partially describes the entity. The description includes the classification of the entity by one or more classifiers of which the entity is an instance.

In UML, an object is an entity with a well-defined boundary and identity that encapsulates state and behaviour. State is represented by attributes and relationships. The behaviour of UML object mapping to ODP *information objects* is represented by state machines.

### 8.2.2 Information object type

An *information object type* is modelled as a UML class. A class describes a set of objects that share the same specifications of features, constraints, and semantics.

NOTE – The UML concept of class is different to the ODP concept of *class*. A UML class is a "description" of a set of objects, while an ODP *class* is the set of *objects* itself. Therefore, the UML concept of class is closer to the ODP concept of *type*, and there is no UML concept corresponding to the ODP concept of *class*. Therefore, no mapping for the ODP concept of *class* is provided.

### 8.2.3 Information object template

An ODP *object template* is modelled as a UML concrete class (i.e., a class that can be directly instantiated).

### 8.2.4 Information action and action types

In the information viewpoint, *actions* are mainly used for describing events that cause state changes, or for implementing *communications* between *objects*, i.e., flows of information.

In an information specification, an *internal action* is mapped to an internal transition of a state of the state machine for the *information object* concerned.

An *interaction* is mapped to a signal sent or received by the state machines of the *information objects* concerned. An ODP *action type* is then mapped to a UML Signal.

### 8.2.5 Relationships between information objects and between information object types

A relationship between information object types, when modelled as part of the *state* of the *objects* of those *types*, can be mapped to a UML association between the UML classes modelling those *types*. In UML, an association is defined as the semantic relationship between one or more classifiers that specifies connections between their instances.

Instances of UML associations (i.e., links) will model the relationships between the *information objects*.

When associations between *information objects* are modelled in ODP as *invariant schemata*, the mapping rules in Clause 8.2.6 apply.

### 8.2.6 Invariant schema

*Invariant schemata* may impose different kinds of constraints in an information specification.

First, *invariant schemata* can provide the specification of the *types* of one or more *information objects*, that will always be satisfied by whatever *behaviour* the *objects* might exhibit. This kind of *invariant schema* may be represented in a UML Package, and drawn in a class diagram, which specifies a set of *object types* (in terms of the set of UML classes that represent such *object types*), their possible relationships (represented as UML associations), and constraints on those *object types*, on their relationships, and possibly on their *behaviours* (represented by the specification of the corresponding UML objects' state machines). The association multiplicities and the UML constraints on the different modelling elements will constrain the possible states and state changes of the UML elements to which they apply.

NOTE – OCL is the recommended notation for expressing the constraints on the modelling elements that form part of the UML representation of an *invariant schema*. However, other notations can be used when OCL does not provide enough expressive power, or is not appropriate due to the kind of expected user of the specification. For example, a temporal logic formula or an English text can be used for expressing a constraint that imposes some kind of fairness requirement on the *behaviour* of the *system* (e.g., "Objects of class X will produce requests to objects of class Y, no later than a given time T after condition A on objects of classes X, Y and Z is satisfied").

As noted in ODP there are cases, however, in which an *invariant schema* in an information viewpoint specification is defined over a set of concrete *information objects*. Such a kind of *invariant schema* may be represented as a UML package of UML objects. The UML constraints on these objects, together with the specifications of the UML classifiers of these objects, constrain the possible states and state changes of the UML objects.

NOTE – The UML classifiers of the objects will constrain the possible states and state changes of the UML objects to which they apply (through the UML associations, state machines, and constraints of these classifiers).

Finally, individual UML constraints can also be used to capture *invariant schemata*.

### 8.2.7 Static schema

An ODP *static schema* is represented as a UML package of UML objects, their attribute links, their UML link ends which have an associated target link end which is navigable, and their UML classifiers.

NOTE – The possible associations of the *information objects* described in a *static schema* with other *objects* not contemplated in the schema need not be included in the UML package, since they are not part of the specification provided by the *schema*. Therefore, whenever the absence of an association instance (i.e., a link) needs to be expressed, it should be explicitly stated (by, e.g., using constraints attached to the appropriate objects).

### 8.2.8 Dynamic schema

A *dynamic schema* is expressed in terms of state machines for the *information objects* in the information specification. The *actions* that relate to the *state* changes are mapped to signals that are sent and received on transitions of the state machines.

### 8.2.9 Summary of the UML mappings for the information language

The Information language profile (IV_Profile) specifies how the information viewpoint modelling concepts relate to and are represented in standard UML using stereotypes, tag definitions, and constraints. It represents the concepts of the information language model (see [8.1.11]).

The following Figure shows the graphical representation of the UML Profile for the Information Language, using the notation provided by UML 2.0.

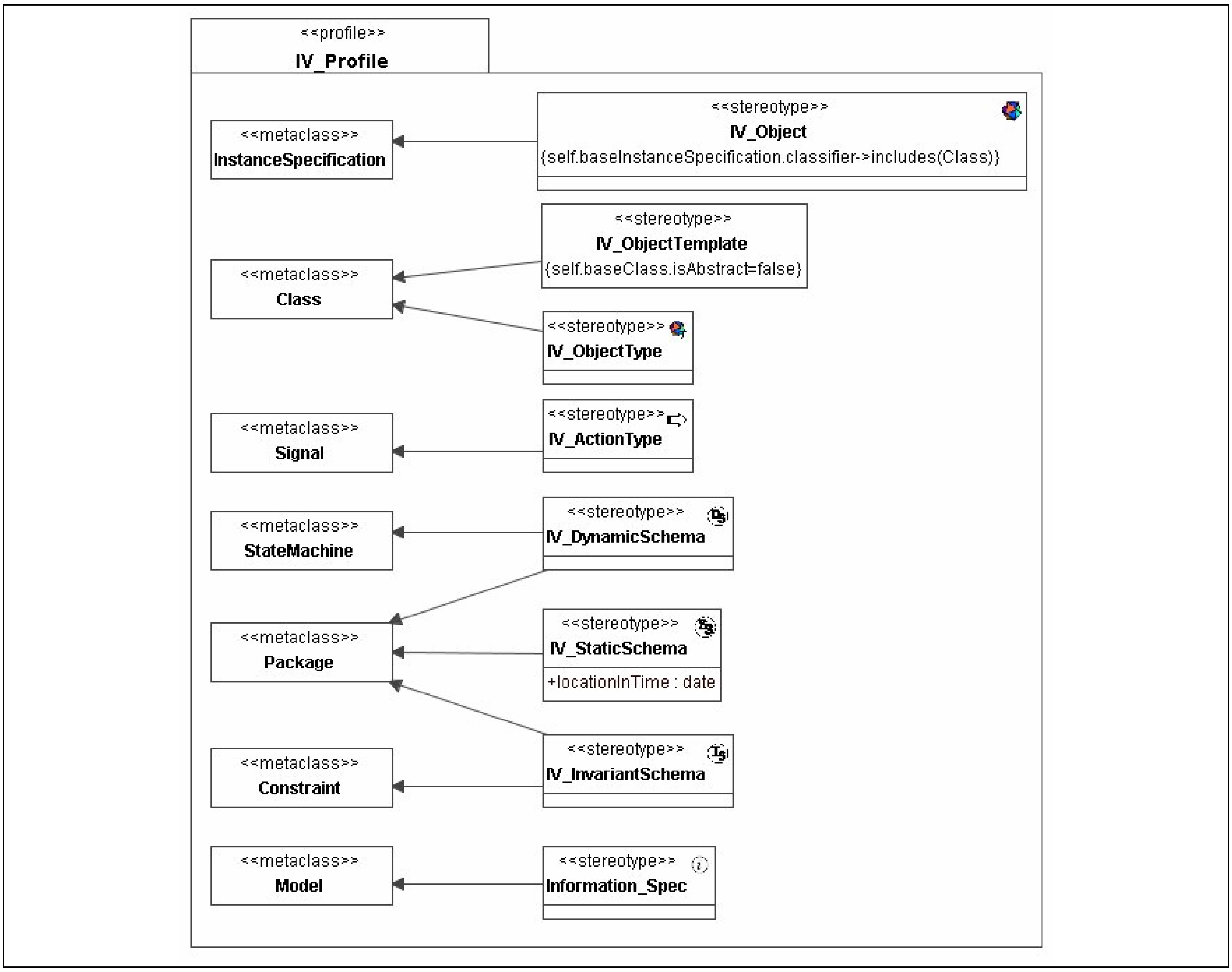



**Figure 17 – Graphical representation of the Information Language profile**

The constraint on the InstanceSpecification element stereotyped as «IV_Object» states that it should be an UML object, i.e., an instance of a Class. The constraint on the Class element stereotyped as «IV_ObjectTemplate» states that a Class that maps to an *information object template* should be a concrete class, i.e., a class that contains all the information to instantiate objects.

## 8.3 Information specification structure (in UML terms)

All the UML elements representing the information specification will be defined within a UML model, stereotyped «Information_Spec». Such a model contains the UML packages that represent the *invariant*, *static* and *dynamic schemata* of the system.

These packages may be defined and organized as follows:

– In the first place, a set of «IV_InvariantSchema» packages with UML class diagrams will define the *information object* and *object types* of the system, their relationships, and the constraints on these elements.

– Second, a set of «IV_StaticSchema» packages with UML object diagrams will represent the state of the system or parts of it at specific locations in time — that may be of interest to any of the system stakeholders. The classifiers of the objects of these diagrams should have been previously defined in the «IV_InvariantSchema» packages that define the structure and composition of the system.

– Third, *dynamic schemata* mapping to individual state machines, will be associated with the corresponding elements in the previous packages. Thus, individual state machines will be associated with the corresponding object types or objects. Likewise, constraints describing invariants and pre- and post-conditions of signals will be associated to the states of the state machines and with the corresponding object type definitions.

– Finally, a set of «IV_InvariantSchema» constraints will impose further constraints on the elements of all the previous packages. Such constraints can be either directly attached to the corresponding UML elements, establishing an implicit context by attachment, or they can form part of a separate piece of specification in which the context of each constraint is explicitly established by naming.

## 8.4 Viewpoint correspondences for the information language

### 8.4.1 Enterprise and information viewpoint specification correspondences

In general, not all the elements of the enterprise specification of a system need to correspond to elements of its information specification. However, the information viewpoint shall conform to the *policies* of the enterprise viewpoint and, likewise, all enterprise *policies* shall be consistent with the *static*, *dynamic*, *invariant schemata* of the information specification.

Where there is a correspondence between information and enterprise elements (e.g., between an *enterprise object* and the *information object* that stores the relevant information about it), the specifier shall provide:

– for each *enterprise object* and for each *artefact role* in an enterprise *action*, the corresponding configuration of *information objects* (if any) that represent them in the information viewpoint;

– for each enterprise *role*, *action* and *process* in the enterprise viewpoint, the corresponding *dynamic* and *invariant schema* definitions in the information viewpoint that specify that *behaviour*.

– for each enterprise *policy* in the enterprise viewpoint, the constraints in the corresponding *schemata* that implement it—since enterprise *policies* may become constraints in any of the *schemata*.

NOTE – In the case of a notional incremental development process of the ODP viewpoint specifications, whereby the information specifications are developed taking into account the previously defined enterprise specifications, *information objects* may be discovered through examination of an enterprise specification. For example, each *artefact* referenced in any *actions* in which an *ODP System* participates will correspond in some way with one or more *information objects*.

### 8.4.2 Information and computational viewpoint specification correspondences

Not all the elements of the information specification of a system need to correspond to elements of its computational specification. In particular, not all states of an information specification need to correspond to states of a computational specification. There may exist transitional computational states within pieces of computational behaviour that are abstracted as atomic transitions in the information specification.

Where an *information object* corresponds to a set of *computational objects*, *static* and *invariant schemata* of the *information object* correspond to possible *states* of the *computational objects*. Every change in state of an *information object* corresponds either to some set of *interactions* between *computational objects*, or to an *internal action* of a *computational object*. The *invariant* and *dynamic schemata* of the *information object* correspond to the *behaviour* and *environment contract* of the *computational objects*.

#### 8.4.3 Information and technology viewpoint specification correspondences

While there may be specific correspondences between information *schemata* and technology viewpoint specifications that require the use of particular technologies, there are neither required correspondences nor required correspondence statements.

NOTE – There may be cases where part of an information viewpoint specification has a direct relationship with a technology viewpoint specification or a choice of technology. Such examples include *invariant schemata* covering performance (e.g., response time) or security.

# 9 Computational Specification

## 9.1 Modelling concepts

The modelling concepts used in a computational specification are defined, together with the structuring rules for their use, in Clause 7 of Part 3 of RM-ODP. Some of the concepts in Part 2 of RM-ODP are also used when defining the computational language concepts. The explanations of the concepts in the text that follows are not normative, and in case of conflicts between these explanations and the text of Parts 2 or 3, the latter documents should be followed.

### 9.1.1 Computational object

An *object* is a model of an entity. An *object* is characterized by its *behaviour* and, dually, by its *state*. An *object* is distinct from any other *object*. An *object* is encapsulated, i.e. any change in its *state* can only occur as a result of an *internal action* or as a result of an *interaction* with its *environment*.

A *computational object* is an *object* as seen in the computational viewpoint. It represents functional decomposition and interacts with other *computational objects*. Since it is an *object*, it has *state* and *behaviour*, and *interactions* are achieved through *interfaces*.

### 9.1.2 Interface [Part 2 – 8.4]

An *interface* is an abstraction of the *behaviour* of an *object* that consists of a subset of the *interactions* of that *object* together with a set of constraints on when they can occur.

### 9.1.3 Interaction [Part 2 – 8.3]

An *interaction* is one of two defined kinds of *actions*. *Action* itself is defined as something that happens, and every *action* of interest for modelling purposes is associated with at least one *object*. The set of *actions* associated with an *object* is partitioned into *internal actions* and *interactions*. An *internal action* always takes place without the participation of the environment of the *object*. An *interaction* takes place with the participation of the environment of the *object*.

### 9.1.4 Environment contract [Part 2 – 11.2.3]

*Environment contract* is a contract between an *object* and its *environment*, including Quality of Service (QoS) constraints, usage and management constraints.

QoS constraints include:

– temporal constraints (e.g. deadlines);

– volume constraints (e.g. throughput);

– dependency constraints covering aspects of availability, reliability, maintainability, security and safety (e.g. mean time between failures).

QoS constraints can imply usage and management constraints. For instance, some QoS constraints (e.g. availability) are satisfied by provision of one or more distribution transparencies (e.g. replication).

An *environment contract* can describe both:

– requirements placed on an *object*'s *environment* for the correct *behaviour* of the *object*, and

– constraints on the *object behaviour* in a correct *environment*.

### 9.1.5 Behaviour (of an object) [Part 2 – 8.6]

*Behaviour of an object* is a collection of *actions* with a set of constraints on when they may occur.

The specification language in use determines the constraints which may be expressed. Constraints may include, for example, sequentiality, non-determinism, concurrency or real-time constraints.

*Behaviour* may include internal actions.

The *actions* that actually take place are restricted by the *environment* in which the *object* is placed.

#### 9.1.6 Signal [Part 3 – 7.1.1]

A *signal* is an atomic shared *action* resulting in one-way *communication* from an initiating object to a responding object.

#### 9.1.7 Operation [Part 3 – 7.1.3]

An *operation* is an *interaction* between a client *object* and a server *object* which is either an *interrogation* or an *announcement*.

#### 9.1.8 Announcement [Part 3 – 7.1.3]

An *interaction* — the *invocation* — initiated by a client *object* resulting in the conveyance of information from that client *object* to a server *object*, requesting a function to be performed by that server *object*.

#### 9.1.9 Interrogation [Part 3 – 7.1.4]

An *interaction* consisting of:

- one *interaction* — the *invocation* — initiated by a client *object*, resulting in the conveyance of information from that client *object* to a server *object*, requesting a function to be performed by the server *object*,

followed by

- a second *interaction* — the *termination* — initiated by the server *object*, resulting in the conveyance of information from the server *object* to the client *object* in response to the invocation.

#### 9.1.10 Flow [Part 3 – 7.1.5]

A *flow* is an abstraction of a sequence of *interactions*, resulting in conveyance of information from a producer *object* to a consumer *object*.

#### 9.1.11 Signal interface [Part 3 – 7.1.6]

A *signal interface* is an *interface* in which all the *interactions* are *signals*.

#### 9.1.12 Operation interface [Part 3 – 7.1.7]

An *operation interface* is an *interface* in which all the *interactions* are *operations*.

#### 9.1.13 Stream interface [Part 3 – 7.1.4]

A *stream interface* is an *interface* in which all the *interactions* are *flows*.

#### 9.1.14 Computational object template [Part 3 – 7.1.9]

A *computational object template* is an *object template* which comprises a set of computational *interface templates* that the object can instantiate, a *behaviour* specification and a *environment contract* specification.

#### 9.1.15 Computational interface template [Part 3 – 7.1.9]

A *computational interface template* is an *interface template* for either a *signal interface*, a *stream interface* or an *operation interface*. A *computational interface template* comprises a *signal*, a *stream* or an *operation interface signature* as appropriate, a *behaviour* specification and *environment contract* specification.

#### 9.1.16 Signal interface signature [Part 3 – 7.1.11]

A *signal interface signature* is an *interface signature* for a *signal interface*. A *signal interface signature* comprises a finite set of *action templates*, one for each *signal type* in the *interface*. Each *action template* comprises the name for the *signal*, the number, names and types of its parameters and an indication of causality (initiating or responding, but not both) with respect to the *object* that instantiates the *template*.

### 9.1.17 Operation interface signature [Part 3 – 7.1.12]

An *operation interface signature* is an *interface signature* for an *operation interface*. An *operation interface signature* comprises a set of *announcement* and *interrogation signatures* as appropriate, one for each *operation type* in the *interface*, together with an indication of causality (client or server, but not both) for the *interface* as a whole, with respect to the *object* which instantiates the *template*.

Each *announcement signature* is an *action template* containing the name of the *invocation* and the number, names and types of its parameters.

Each *interrogation signature* comprises an *action template* with the following elements:

– the name of the *invocation*;

– the number, names and types of its parameters,

– a finite, non-empty set of *action templates*, one for each possible *termination type* of the *invocation*, each containing both the name of the *termination* and the number, names and types of its parameters.

### 9.1.18 Stream interface signature [Part 3 – 7.1.13]

A *stream interface signature* is an *interface signature* for a *stream interface*. A *stream interface* comprises a finite set of *action templates*, one for each *flow type* in the *stream interface*. Each *action template* for a *flow* contains the name of the *flow*, the information *type* of the *flow*, and an indication of causality for the *flow* (i.e. producer or consumer but not both) with respect to the object which instantiates the *template*.

### 9.1.19 Binding object [Part 3 – 7.1.14]

A *binding object* is a *computational object* which supports a *binding* between a set of other c*omputational objects*.

### 9.1.20 Structure of a computational specification

In ODP, a computational specification describes the functional decomposition of an *ODP system*, in distribution transparent terms, as:

– a configuration of *computational objects*;

– the *internal actions* of those *objects*;

– the *interactions* that occur among those *objects*;

– *environment contracts* for those *objects* and their *interfaces*.

The set of *computational objects* specified by the computational specification constitute a configuration that will change as the *computational objects* instantiate further *computational objects* or *computational interfaces*; perform *binding actions*; effect control functions upon *binding objects*; delete *computational interfaces*; or delete *computational objects*.

The computational language defines a set of rules that constrain a computational specification. These comprise:

– *interaction* rules, *binding* rules and *type* rules that provide distribution transparent interworking;

– *template* rules that apply to all *computational objects* and *computational interfaces*;

– *failure* rules that apply to all *computational objects* and identify the potential points of *failure* in computational activities.

### 9.1.21 Model of the computational language

Figure 18 illustrates the concepts of the computational language and the relationships between them.

NOTE – Some of the relationships between Computational language concepts are not shown in Figure 18, e.g. relationship between interface and signature, since they are related through their super-types.

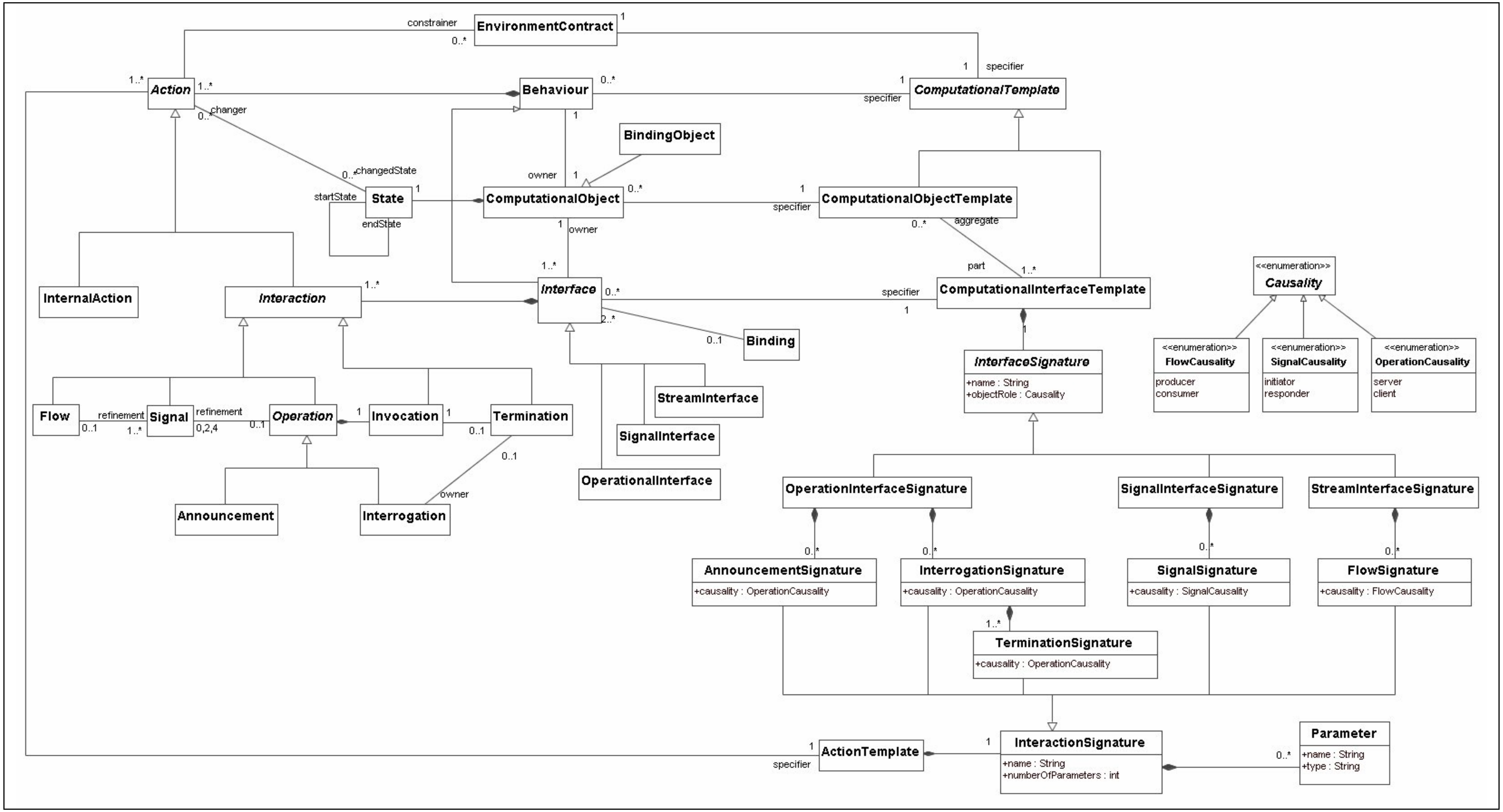



**Figure 18 – Computational language concepts**

The following restrictions apply to the elements of the diagram shown in Figure 18.

- A *binding object* is associated with at least two different *objects*.
- A *binding object* binds two or more *objects* through the same type of *interface* (*signal*, *announcement*, *interrogation*, or *flow*).
- All *interfaces* associated to a *signal interface signature* are *signal interfaces* [9.2.11.3], and all its constituent *interaction signatures* are *signal signatures*.

  **context** Signal **inv** SignalSignature: self.interface->forAll(oclIsTypeOf(SignalInterface))

  **context** SignalInterface **inv** SignalSignature: self.specifier->forAll(oclIsTypeOf(SignalSignature))

  **context** SignalInterface **inv** SignalInterfaceSignature:
  self.specifier->forAll(oclIsTypeOf(SignalInterfaceSignature))

- All *interfaces* associated to an *operation interface signature* are *operation interfaces* [9.2.12.3], and all its constituent *interaction signatures* are *announcement*, *interrogation*, *invocation* or *termination signatures*.

  **context** Announcement **inv** AnnouncementSignature:
  self.interface->forAll(oclIsTypeOf(OperationInterface))

  **context** Invocation **inv** InvocatonSignature: self.interface->forAll(oclIsTypeOf(OperationInterface))

  **context** Termination **inv** TerminationSignature:
  self.interface->forAll(oclIsTypeOf(OperationInterface))

  **context** OperationInterface **inv** OperationInterfaceSignature:
  self.specifier->forAll(oclIsTypeOf(OperationInterfaceSignature))

- All *interfaces* associated to a *stream interface signature* are *stream interfaces* [9.2.13.3].

  **context** Flow **inv** StreamSignature: self.interface->forAll(oclIsTypeOf(StreamInterface))

  **context** StreamInterface **inv** StreamInterfaceSignature:
  self.specifier->forAll(oclIsTypeOf(StreamInterfaceSignature))

## 9.2 UML mappings

The following paragraphs describe how the ODP computational concepts described in the previous Clause are represented in UML in a computational specification. A brief explanation of the UML concepts used in the representation of each concept is given, together with a justification of the representation used.

NOTE – In this clause mappings are only defined for those concepts for which use has been demonstrated through an example, included in the main body of this document or in its annexes. Where no example has been identified, the concept concern is mentioned, but no mapping is offered.

### 9.2.1 Computational object template

A *computational object template* is modelled as a UML component. A UML component represents a modular part of a system which encapsulates its contents, and defines its behaviour in terms of provided and required interfaces through its ports.

The attribute isIndirectlyInstantiated of the component stereotyped «CV_ObjectTemplate» should be set to true. This attribute constraints the he kind of instantiation that applies to a UML component. If false, the component is instantiated as an addressable object. If true (default vuale), the component is defined at design-time, but at runtime (or execution-time) an object specified by the component does not exist, that is, the component is instantiated indirectly, through the instantiation of its realizing classifiers or parts.

### 9.2.2 Computational object

A *computational object* is modelled as a UML InstanceSpecification of component, stereotyped as «CV_Object», since it is an instantiation of a *computational object template*. An InstanceSpecification of component is an instance of a UML classifier component.

### 9.2.3 Binding object

A *binding object* is a kind of computational object, and is modelled as a UML InstanceSpecification of component, stereotyped as «CV_BindingObject».

The following two restrictions apply to *binding objects*, and therefore to components stereotyped «CV_BindingObject»:

- A *binding object* is associated with at least two different *objects*.

— A *binding object* binds two or more *objects* through the same *type* of *interface* (*signal*, *announcement*, *interrogation*, or *flow*).

#### 9.2.4 Environment contract

An *environment contract* is modelled as a UML package, stereotyped as «CV_EnvironmentContract», when representing a set of structural and behavioural constraints between a *computational object* and its *environment*, including quality of service and other kinds of requirements. In addition, individual UML constraints applied to UML model elements can also be stereotyped «CV_EnvironmentContract» when they capture such kinds of restrictions.

#### 9.2.5 Signal

A *signal* is modelled as a UML message, stereotyped as «CV_Signal», sent by an *initiating object* and received by a *responding object*.

#### 9.2.6 Announcement

An *announcement* is modelled as a UML message, stereotyped as «CV_Announcement», sent by a *client object* and received by a *server object* with no response expected.

#### 9.2.7 Invocation

An *invocation* is a part of *interrogation* and is modelled as a UML message, stereotyped as «CV_Invocation», sent by a *client object* and received by a *server object*.

#### 9.2.8 Termination

A *termination* is a part of *interrogation* and is modelled as a UML message, stereotyped as «CV_Termination», sent by a *server object* and received by a *client object*.

#### 9.2.9 Flow

A *flow* is modelled as a UML message as well as a UML interaction. It is modelled as a UML message sent by a *producer object* and received by a *consumer object*, stereotyped as «CV_Flow». It is also a UML interaction between a *producer object* and a *consumer object* for the message transfer, stereotyped as «CV_Flow».

#### 9.2.10 Computational interface

An *interface* of a *computational object* is modelled as a port of a UML component instance. A port is an interaction point between a UML component and its environment or between a UML component and its internal parts (UML components). A port supports two types of interfaces: provided interfaces and required interfaces.

*Computational interface templates* will be mapped to UML ports, in this case of the UML components that represent the *computational object templates* from which the *objects* are instantiated. Thus, ports of component instances will be stereotyped «CV_Interface», whilst ports of UML component will be stereotyped «CV_InterfaceTemplates»

#### 9.2.11 Signal interface

A *signal interface* is modelled as a port of a UML component instance, stereotyped as «CV_SignalInterface». Through this port, a *computational object* can provide or require a set of *signal interface signatures*.

#### 9.2.12 Operation interface

An *operation interface* is modelled as a port of a UML component, stereotyped as «CV_OperationInterface». Through this port, a *computational object* can provide or require a set of *operation interface signatures*.

#### 9.2.13 Stream interface

A *stream interface* is modelled as a port of a UML component, stereotyped as «CV_StreamInterface». Through this port, a *computational object* can provide or require a set of *stream interface signatures*.

#### 9.2.14 Computational interface signature

A *computational interface signature* is modelled as a UML interface.

#### 9.2.15 Signal interface signature

A *signal interface signature* is modelled as a UML interface, stereotyped as «CV_SignalInterfaceSignature».

#### 9.2.16 Operation interface signature

An *operation interface signature* is modelled as a UML interface, stereotyped as «CV_OperationInterfaceSignature».

### 9.2.17 Stream interface signature

A *stream interface signature* is modelled as a UML interface, stereotyped as «CV_StreamInterfaceSignature».

### 9.2.18 Computational signature

A *Computational signature* can be modelled as a UML reception, UML operation, or interface, depending on the sort of signature. UML receptions will be used to specify *signatures* of *computational interactions* which are expressed as individual *signals* (*signals*, *announcements*, *invocations* and *terminations*). UML operations can be used to map ODP *interrogation signatures* that are composed of an *invocation signature* and a *termination signature*. Finally, UML interfaces will be used for mapping *flow signatures*, when *flows* are expressed in terms of sequences of *signals*.

### 9.2.19 Signal signature

A *signal signature* is modelled as a UML reception, stereotyped as «CV_SignalSignature». This stereotyped UML reception represents an *action template* which includes name for the signal, the number, names and types of its parameters, and indication of initiating or responding.

### 9.2.20 Announcement signature

An *announcement signature* is a signature for *announcement*. An *announcement signature* is modelled as a UML reception, stereotyped as «CV_AnnouncementSignature». This stereotyped UML interface represents an *action template* which includes name for the invocation, the number, names and types of its parameters, and indication of client or server.

### 9.2.21 Invocation signature

An *invocation signature* is a signature for an *invocation* in an *interrogation*. An *invocation signature* is modelled as a UML reception, stereotyped as «CV_InvocationSignature». This stereotyped UML reception represents an *action template* which includes name for the *invocation*, the number, names and types of its parameters, and indication of client or server.

### 9.2.22 Termination signature

A *termination signature* is a signature for a *termination* for *interrogation*. A *termination signature* is modelled as a UML reception, stereotyped as «CV_TerminationSignature». This stereotyped UML reception represents an *action template* which includes name for the *termination*, the number, names and types of its parameters, and indication of client or server.

### 9.2.23 Interrogation signature

An *interrogation signature* is a *signature* for an *interrogation*, which comprises signatures for an *invocation* and a *termination*. An *interrogation signature* is modelled as a UML operation, stereotyped as «CV_InterrogationSignature». This stereotyped UML operation represents an *action template* which includes name for the *invocation*, the number, names and types of its parameters, the indication of client or server, and the number, names and types of the *termination*'s parameters.

Alternatively, an *interrogation signature* can be specified in terms of separated *invocation* [9.2.21] and *termination signatures* [9.2.22].

### 9.2.24 Flow signature

A *flow signature* is modelled as a UML interface, stereotyped as «CV_FlowSignature». This stereotyped UML interface represents an *action template* which includes name for the flow, the number, names and types of its signals, their parameters, and indication of producer or consumer.

### 9.2.25 Summary of the UML mappings for the computational language

The Computational language profile (CV_Profile) specifies how the computational viewpoint modelling concepts relate to and are represented in standard UML using stereotypes, tag definitions, and constraints. It represents the concepts of the computational language model (see [9.1.3]).

The following shows diagrammatic representations of this UML profile.

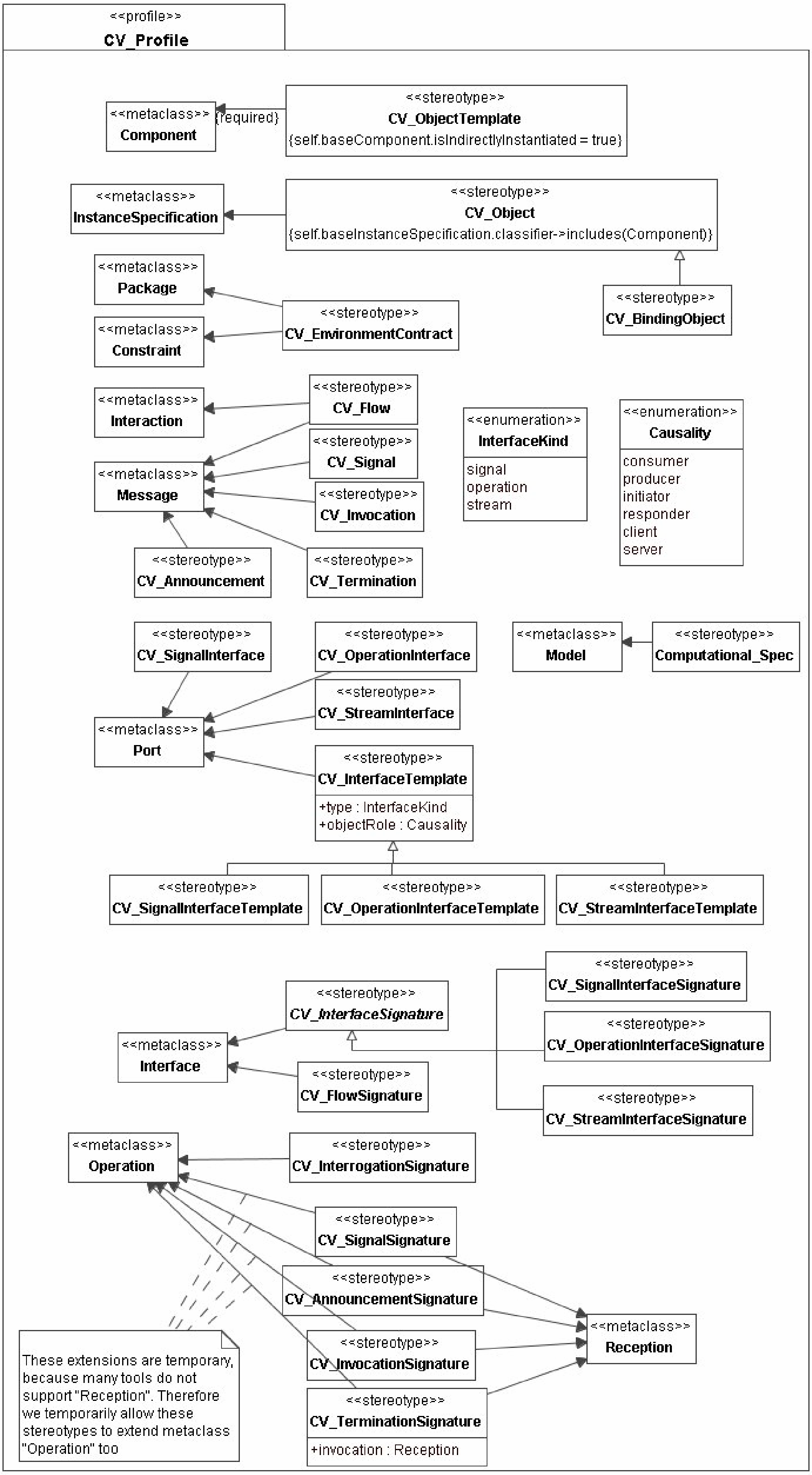



**Figure 19 – Graphical representation of the Computational Language profile (using the UML 2.0 notation)**

The following restrictions apply to the elements of the Profile shown in Figure 19:

- The constraint baseComponent.isIndirectlyInstantiated=true means that the component is defined at design-time, but at runtime (or execution-time) an object specified by the component does not exist, that is, the component is instantiated indirectly, through the instantiation of its realizing classifiers or parts.
- A component representing a *computational object template* has ports and interfaces for *interaction* with other *computational objects*.

In addition, the elements of the computational language (shown in Figure 18) were subject to a set of restrictions, as described in [9.1.21]. The constraints that implement those restrictions on the corresponding profile elements should also apply.

## 9.3 Computational specification structure (in UML terms)

All the UML elements representing the Computational specification will be defined within a UML model, stereotyped «Computational_Spec». Such a model contains UML packages that represent:

- a configuration of *computational objects* with dependencies among those *objects* using required and provided interfaces and signatures they provide, with UML component diagram,
- structure of *computational objects* including composition and decomposition of *computational objects*, with UML component diagram,
- *environment contract* for *computational objects*, with UML constraints on UML model elements,
- *interactions* between *computational objects*, and *interactions* between *composed computational objects* within a *computational object*, with UML activity diagrams, state charts, and interaction diagrams.

## 9.4 Viewpoint correspondences for the computational viewpoint

### 9.4.1 Enterprise and computational viewpoint specification correspondences

The specifier shall provide:

- for each *enterprise object* in the enterprise specification, that configuration of *computational objects* (if any) that realizes the required *behaviour*;
- for each *interaction* in the enterprise specification, a list of those *computational interfaces* and *operations* or *streams* (if any) that correspond to the enterprise *interaction*, together with a statement of whether this correspondence applies to all occurrences of the *interaction*, or is qualified by a predicate;
- for each *role* affected by a *policy* in the enterprise specification, a list of the *computational object types* (if any) that exhibit choices in the *computational behaviour* that are modified by the *policy*;
- for each *interaction* between *roles* in the enterprise specification, a list of *computational binding object types* (if any) that are constrained by the enterprise *interaction*;
- for each enterprise *interaction type*, a list of *computational behaviour types* (if any) capable of representing (i.e. acting as a carrier for) the enterprise *interaction type*.

If a process based approach is taken, the specifier shall provide:

- for each *step* in the *process*, a list of participating *computational objects* which may fulfil one or more of *actor roles*, *artefact roles*, and *resource roles*.

Temporary Note – NBs are requested to provide proposals for how to use UML to represent, and if possible, police these statements.

### 9.4.2 Information and computational viewpoint specification correspondences

This document does not prescribe exact correspondences between *information objects* and *computational objects*. In particular, not all *states* of a computational specification need to correspond to *states* of an information specification. There may exist transitional computational *states* within pieces of *computational behaviour* that are abstracted as atomic transitions in the information specification.

Where an *information object* corresponds to a set of *computational objects*, *static* and *invariant schemata* of an *information object* correspond to possible *states* of the *computational objects.* Every change in *state* of an *information object* corresponds either to some set of *interactions* between *computational objects* or to an *internal action* of a *computational object*. The *invariant* and *dynamic schemata* of the *information object* correspond to the *behaviour* and *environment contract* of the *computational objects*.

#### 9.4.4 Computational and engineering viewpoint specification correspondences

Each *computational object* that is not a *binding object* corresponds to a set of one or more *basic engineering objects* (and any *channels* which connect them). All the *basic engineering objects* in the set correspond only to that *computational object*.

Except where transparencies which replicate objects are involved, each *computational interface* corresponds exactly to one *engineering interface*, and that *engineering interface* corresponds only to that *computational interface*.

> NOTE – The *engineering interface* is supported by one of the *basic engineering objects* that corresponds to the *computational object* supporting the *computational interface*.

Where transparencies that replicate *objects* are involved, each *computational interface* of the *objects* being replicated correspond to a set of *engineering interfaces*, one for each of the *basic engineering objects* resulting from the replication. These *engineering interfaces* each correspond only to the original *computational interface*.

Each *computational interface* is identified by any member of a set of one or more *computational interface* identifiers. Each *engineering interface* is identified by any member of a set of one or more *engineering interface* references. Thus, since a *computational interface* corresponds to an *engineering interface*, an identifier for a *computational interface* can be represented unambiguously by an *engineering interface* reference from the corresponding set.

Each *computational binding* (either *primitive bindings* or *compound bindings* with associated *binding objects*) corresponds to either an *engineering local binding* or an *engineering channel*. This *engineering local binding* or *channel* corresponds only to that *computational binding*. If the *computational binding* supports *operations*, the *engineering local binding* or *channel* shall support the interchange of at least

- *computational signature* names;
- *computational operation* names;
- *computational termination* names;
- *invocation* and *termination* parameters (including *computational interface* identifiers and *computational interface signatures*).

Except where transparencies that replicate *objects* are involved, each *computational binding object control interface* has a corresponding *engineering interface* and there exists a chain of *engineering interactions* linking that *interface* to any *stubs*, *binders*, *protocol objects* or *interceptors* to be controlled in support of the *computational binding*.

> NOTE – The set of *control interfaces* involved depends on the *type* of the *binding object*.

Each *computational interaction* corresponds to some chain of *engineering interactions*, starting and ending with an *interaction* involving one or more of the *basic engineering objects* corresponding to the interacting *computational objects*.

Each *computational signal* corresponds either to an *interaction* at an *engineering local binding* or to a chain of *engineering interactions* that provides the necessary consistent view of the *computational interaction*.

The transparency prescriptions in [Part 3 – 16] specify additional correspondences.

> NOTE 1 – *Basic engineering objects* corresponding to different *computational objects* can be members of the same *cluster*.

> NOTE 2 – In an entirely object-based computational language, data are represented as abstract data types (i.e., *interfaces* to *computational objects*).

> NOTE 3 – *Computational interface* parameters (including those for abstract data types) can be passed by reference, such parameters correspond to *engineering interface* references.

> NOTE 4 – *Computational interface* parameters (including those for abstract data types) can be passed by migrating or replicating the *object* supporting the *interface*. In the case of migration such parameters correspond to *cluster templates*.

> NOTE 5 – If the abstract *state* of a *computational object* supporting an *interface* parameter is invariant, the *object* can be cloned rather than migrated.

> NOTE 6 – *Cluster templates* can be represented as abstract data types. Thus strict correspondences between computational parameters and *engineering interface* references are sufficient. The use of *cluster templates* or data are important engineering optimisations and therefore not excluded.

> Temporary Note – NBs are requested to provide proposals for how to use UML to represent, and if possible, police these statements.

# 10 Engineering Specification

## 10.1 Modelling concepts

The modelling concepts used in an engineering specification are defined, together with the structuring rules for their use, in Clause 8 of Part 3 of RM-ODP. The explanations of the concepts in the text that follows are not normative, and in case of conflicts between these explanations and the text in Part 3, the latter should be followed.

### 10.1.1 Basic concepts

#### 10.1.1.1 Basic engineering object

A *basic engineering object* is an *engineering object* that requires the support of a distributed infrastructure.

#### 10.1.1.2 Cluster

A *cluster* is a configuration of *basic engineering objects* forming a single unit for the purposes of *deactivation*, *checkpointing*, *reactivation*, *recovery* and *migration*.

#### 10.1.1.3 Cluster manager

A *cluster manage*r is an *engineering object* that manages the basic *engineering objects* in a *cluster*.

#### 10.1.1.4 Capsule

A *capsule* is a *configuration* of *engineering objects* forming a single unit for the purpose of encapsulation of processing and storage.

#### 10.1.1.5 Capsule manager

A *capsule manager* is an *engineering object* that manages the *engineering objects* in a *capsule*.

#### 10.1.1.6 Nucleus

A *nucleus* is an *engineering object* that coordinates processing, storage and communications functions for use by other *engineering objects* within the *node* to which it belongs.

#### 10.1.1.7 Node

A *node* is a *configuration* of *engineering objects* forming a single unit for the purpose of *location in space*, and that embodies a set of processing, storage and communication functions.

### 10.1.2 Channel concepts

#### 10.1.2.1 Channel

A *channel* is a configuration of *stubs*, *binders*, *protocol objects* and *interceptors* providing a *binding* between a set of *interfaces* to basic *engineering objects*, through which *interaction* can occur.

#### 10.1.2.2 Stub

A *stub* is an *engineering object* in a *channel*, which interprets the *interactions* conveyed by the *channel*, and performs any necessary transformation or monitoring based on this interpretation.

#### 10.1.2.3 Binder

A *binder* is an *engineering object* in a *channel*, which maintains a distributed *binding* between interacting basic *engineering objects*.

#### 10.1.2.4 <X> Interceptor

An <X> *interceptor* is an *engineering object* in a *channel*, placed at a boundary between <X> *domains*. An <X> *interceptor*

– performs checks to enforce or monitor policies on permitted interactions between basic *engineering objects* in different *domains*;

– performs transformations to mask differences in interpretation of data by basic *engineering objects* in different *domains*.

#### 10.1.2.5 Protocol object

A *protocol object* is an *engineering object* in a *channel*, which communicates with other *protocol objects* in the same *channel* to achieve interaction between basic *engineering objects* (possibly in different *clusters*, possibly in different *capsules*, possibly in different *nodes*).

#### 10.1.2.6 Communications domain

A *communication domain* is a set of *protocol objects* capable of interworking.

#### 10.1.2.7 Communication interface

A *communication interface* is an *interface* of a *protocol object* that can be bound to an *interface* of either an *interceptor object* or another *protocol object* at an interworking reference point.

### 10.1.3 Identifier concepts

#### 10.1.3.1 Binding endpoint identifier

A *binding endpoint identifier* is an identifier, in the naming context of a *capsule*, used by a basic *engineering object* to select one of the *bindings* in which it is involved, for the purpose of *interaction*.

#### 10.1.3.2 Engineering interface reference

An *engineering interface reference* is an identifier, in the context of an *engineering interface reference management domain*, for an *engineering object interface* that is available for distributed *binding*.

#### 10.1.3.3 Engineering interface reference management domain

An *engineering interface reference management domain* is a set of *nodes* forming a naming *domain* for the purpose of assigning *engineering interface references.*

#### 10.1.3.4 Engineering interface reference management policy

An *engineering interface reference management policy* is a set of *permissions* and *prohibitions* that govern the *federation* of *engineering interface reference management domains*.

#### 10.1.3.5 Cluster template

A *cluster template* is an *object template* for a configuration of *objects,* with any *activity* required to instantiate those *objects* and establish initial *bindings*.

### 10.1.4 Checkpointing concepts

#### 10.1.4.1 Checkpoint

A *checkpoint* is an *object template* derived from the state and structure of an *engineering object* that can be used to instantiate another *engineering object*, consistent with the state of the original *object* at the time of *checkpointing*.

#### 10.1.4.2 Checkpointing

*Checkpointing* is to create a *checkpoint*. *Checkpoints* can only be created when the *engineering object* involved satisfies a pre-condition stated in a *checkpointing policy*.

#### 10.1.4.3 Cluster checkpoint

A *cluster checkpoint* is a *cluster template* containing *checkpoints* of the basic *engineering objects* in a *cluster*.

#### 10.1.4.4 Deactivation

*Deactivation is to checkpoint* a *cluster*, followed by deletion of the *cluster*.

#### 10.1.4.5 Cloning

Cloning is to instantiate a *cluster* from a *cluster checkpoint.*

#### 10.1.4.6 Recovery

*Recovery* is to clone a *cluster* after *cluster* failure or deletion.

#### 10.1.4.7 Reactivation

*Reactivation* is to clone a *cluster* following its deactivation.

#### 10.1.4.8 Migration

*Migration* is to move a *cluster* to a different *capsule*.

### 10.1.5 Model of the engineering language

#### 10.1.5.1 Logical models and physical deployment models

When modelling systems, it is useful to consider the distinction between logical models and physical models. A logical model describes the logical elements of a system, while a physical model describes physical artefacts and resources deployed at runtime.

A model of the Computational Viewpoint is a logical model. The Engineering Viewpoint refines this logical model to a technology-independent model, e.g. distributed component model and messaging system. These kinds of refined models are technology-independent logical models, for their respective platform styles.

The Engineering Viewpoint diagrams may also be physical deployment models. More specifically, the Engineering Viewpoint diagrams may be technology-independent physical deployment models, and the Technology Viewpoint diagrams may be technology-specific physical deployment models.

Both of these approaches are valid and complementary. Both the Engineering and Technology Viewpoints, therefore, break down into logical and deployment viewpoints, as Figure 20 illustrates.

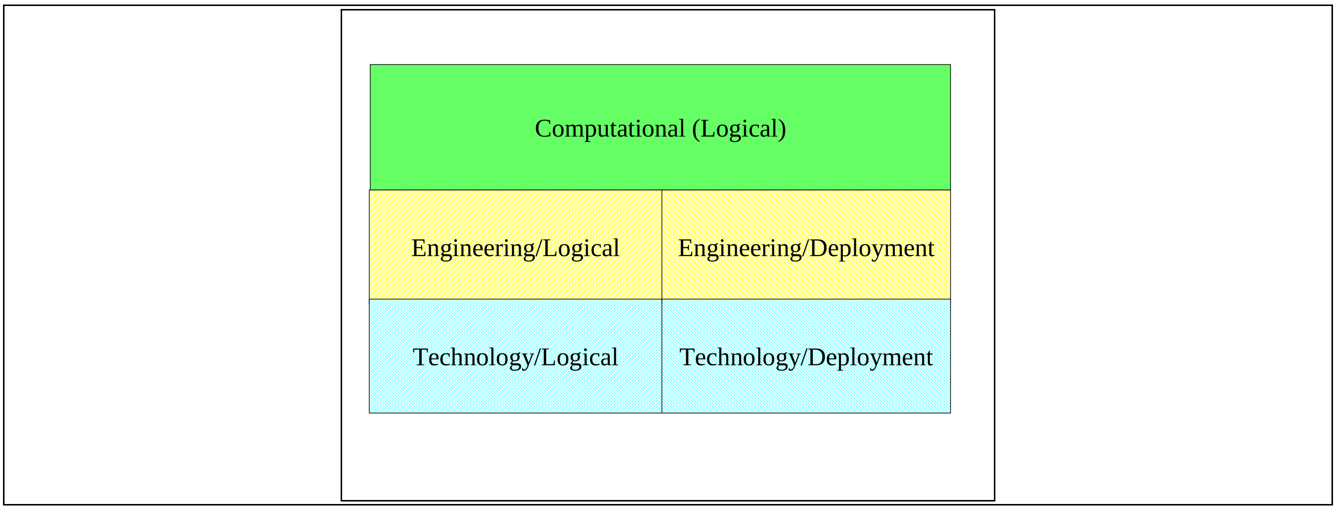



**Figure 20 – Logical and physical viewpoints**

The diagrams below illustrate the concepts of the engineering language and the relationships between them. The model for the Engineering Language is presented here with four partial diagrams.

#### 10.1.5.2 Engineering Objects

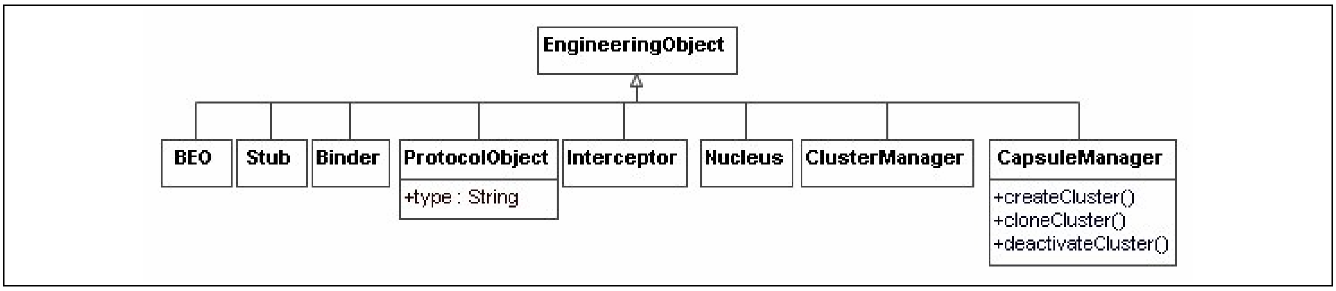



**Figure 21 – Engineering objects**

NOTE – In the Figure, and in the text that follows, *BEO* stands for *Basic Engineering Object*.

#### 10.1.5.3 Node structure

The node structure is about structuring of a node with *nucleus*, *capsule*, *cluster* and various *engineering objects*.

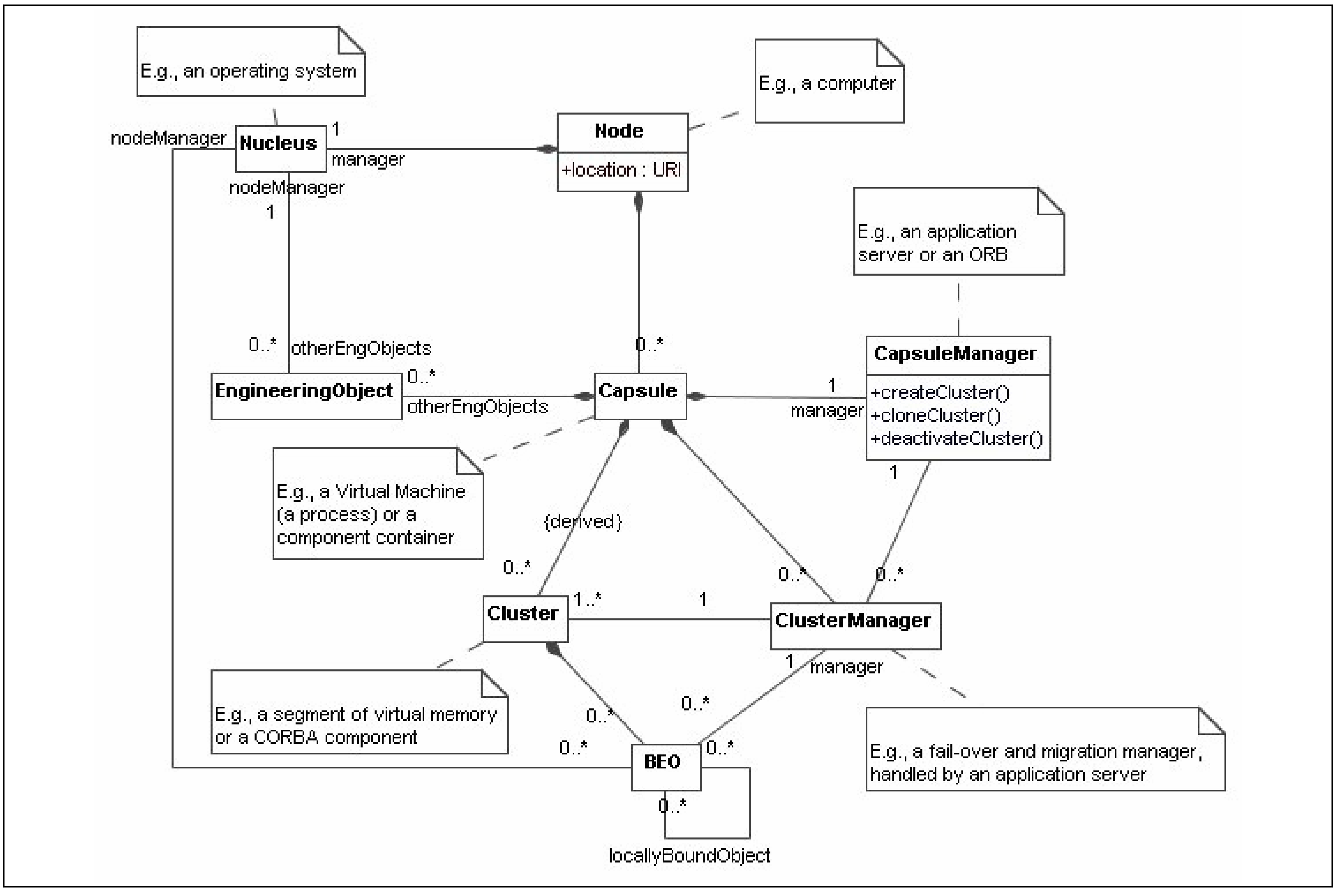



**Figure 22 – Engineering language – basic concepts**

The following constraints apply to the elements of the engineering language shown in Figure 22:

- Each *Stub* to which a *BEO* is related must be part of a *Channel* to which the *BEO* is related

  **context** BEO **inv** SameChannel:
  self.stub->forAll (stub | self.channel->exists (channel | channel = stub.channel))

- For each *Channel* to which a *BEO* is related, the *BEO* must be related to exactly one *Stub* that is part of that *Channel*

  **context** BEO **inv** OneStubPerChannel:
  self.channel->forAll (channel | self.stub->select (stub | stub.channel = channel )->size ( ) = 1 )

- In order for two *BEOs* to be locally bound to each other, they must reside in the same *cluster*

  **context** BEO **inv** SameCluster:
  self.locallyBoundObject->forAll (obj | obj.cluster = self.cluster)

- A *BEO* binds to the *node management interface* provided by the *Nucleus* associated with the *Node* that contains the *Capsule* that contains the *Cluster* that contains the *BEO*

  **context** BEO **inv** NodeManagerDerivationRule:
  self.nodeManager = self.cluster.capsule.node.manager

- The *engineering object's node* manager should be the same as the *node* manager that contains

  **context** EngineeringObject **inv** NodeManagerDerivationRule:
  self.nodeManager = self.capsule.node.manager

- The *Capsule* to which a *Cluster* belongs is the *Capsule* to which the *Cluster's* manager belongs

  **context** Cluster **inv** CapsuleDerivationRule:
  self.capsule = self.manager.capsule

- Derivation Rule: The *CapsuleManager* to which the *ClusterManager* is bound is the *CapsuleManager* of the *Capsule* that contains the *Clusters* that the *CapsuleManager* manages

  **context** ClusterManager **inv** CapsuleManager:
  self.cluster->forAll (c : capsule | c.manager = self.capsuleManager)

– The set of other *engineering objects* that the *Capsule* owns and the set of *ClusterManagers* that the *Capsule* owns are disjoint

**context** Capsule **inv** NoOtherEOisClusterManager:
self.otherEngObject->intersection(self.clusterManager)->isEmpty( )

– The set of other *engineering objects* that the *Capsule* owns and the set of *CapsuleManagers* that the *Capsule* owns are disjoint

**context** Capsule **inv** NoOtherEOisCapsuleManager:
not self.otherEngObject->includes(self.manager)

#### 10.1.5.4 Channels

This part is about communication enabling model elements around *channels*.

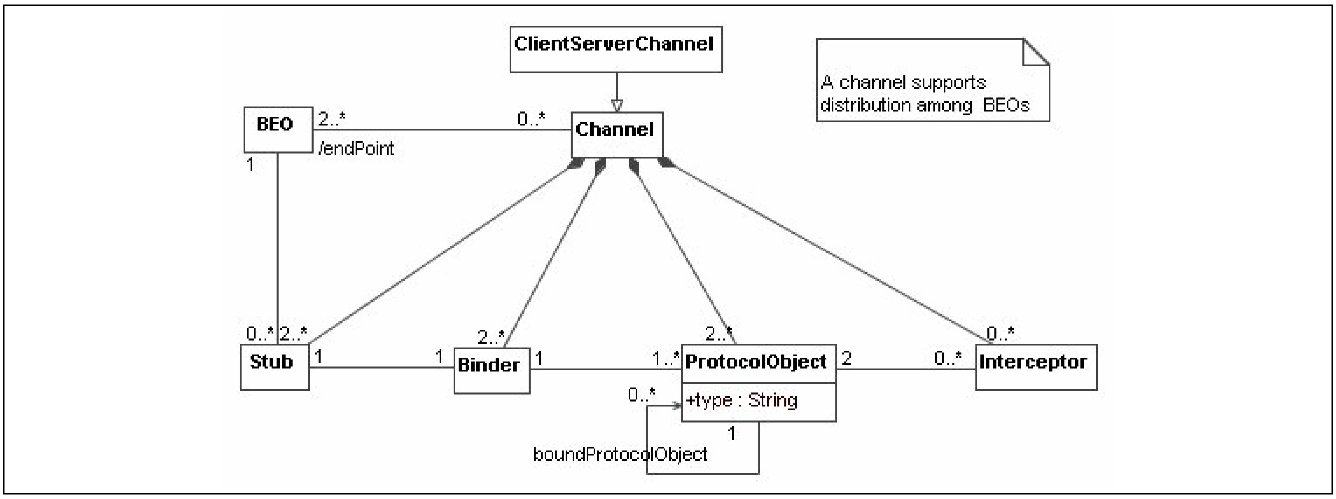



**Figure 23 – Engineering language model – Channels**

The following constraints apply to the concepts expressed in the diagram of Figure 23:

- The collection of *BEOs* that are the end points linked by a *Channel* is derived by adding to the collection, for each *Stub* in the *Channel*, the *BEO* to which the *Stub* is related

  **context** Channel **inv** EndPointDerivationRule:
  self.endPoint->includesAll(self.stub.bEO) and self.stub.bEO->includesAll(self.endPoint)

- The *BEOs* constituting a *Channel's* endpoints must each reside in different *Clusters*

  **context** Channel **inv** EndPointsInDifferentClusters:
  self.endPoint->forAll (ep1, ep2 | ep1.cluster <> ep2.cluster)

- The *BEO* and *Binder* to which a *Stub* is related are parts of the same *Channel* of which the *Stub* is a part

  **context** Stub **inv** SameChannelStub:
  self.bEO.channel = self.channel and self.binder.channel = self.channel

- The *Stub* to which a *Binder* is related and the *ProtocolObjects* to which the *Binder* is related are all parts of the same *Channel* of which the *Binder* is a part

  **context** Binder **inv** SameChannelBinder:
  self.protocolObject->forAll (po | po.channel = self.channel) and self.stub.channel = self.channel

- The *ProtocolObjects* for which an *Interceptor* provides protocol conversion must be part of the same *Channel* of which the *Interceptor* is a part

  **context** Interceptor **inv** SameChannelInterceptor:
  self.protocolObject->forAll (po | po.channel = self.channel)

- Any *Interceptor* to which a *ProtocolObject* is related and the *Binder* to which the *ProtocolObject* is related are part of the same *Channel* of which the *ProtocolObject* is a part

  **context** ProtocolObject **inv** SameChannelPO:
  self.interceptor->forAll (i | i.channel = self.channel) and self.binder.channel = self.channel

- In order for two *ProtocolObjects* to be associated, they must be of the same *type*

**context** ProtocolObject **inv** SameType:
self.boundProtocolObject->forAll (po | po.type = self.type)

#### 10.1.5.5 Domains

This part is about kinds of domains and object membership of domains that make up *domains*.

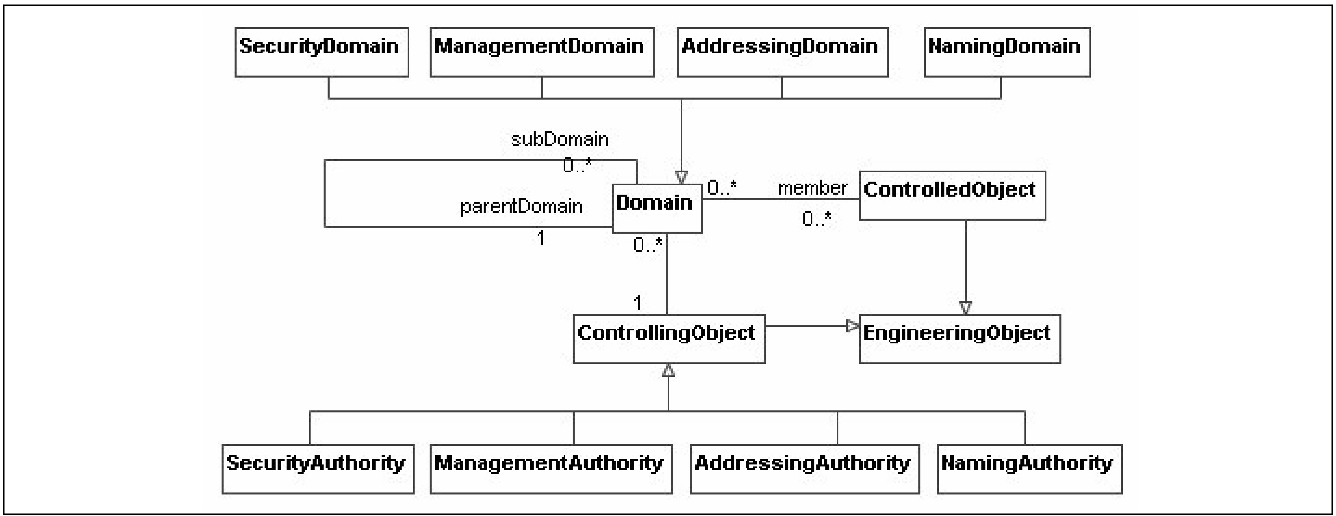



**Figure 24 – Domains**

The following restrictions apply to the model elements depicted in Figure 24:

– All members of a subdomain are members of its parent domain:

**context** Domain **inv** SubDomainIsSubSet:
self.subDomain->forAll (subDomain | self.member->includes(subDomain.member) )

– controlling objects should be associated to the corresponding domains:

**context** SecurityDomain **inv** ControllingObject:
self.controllingObject.oclIsTypeOf(SecurityAuthority)

**context** ManagementDomain **inv** ControllingObject:
self.controllingObject.oclIsTypeOf(ManagementAuthority)

**context** AddressingDomain **inv** ControllingObject:
self.controllingObject.oclIsTypeOf(AddressingAuthority)

**context** NamingDomain **inv** ControllingObject: self.controllingObject.oclIsTypeOf(NamingAuthority)

#### 10.1.5.6 Identifiers

This part is mainly about identity, domain and policy management, with respect to *nodes* and *objects*.

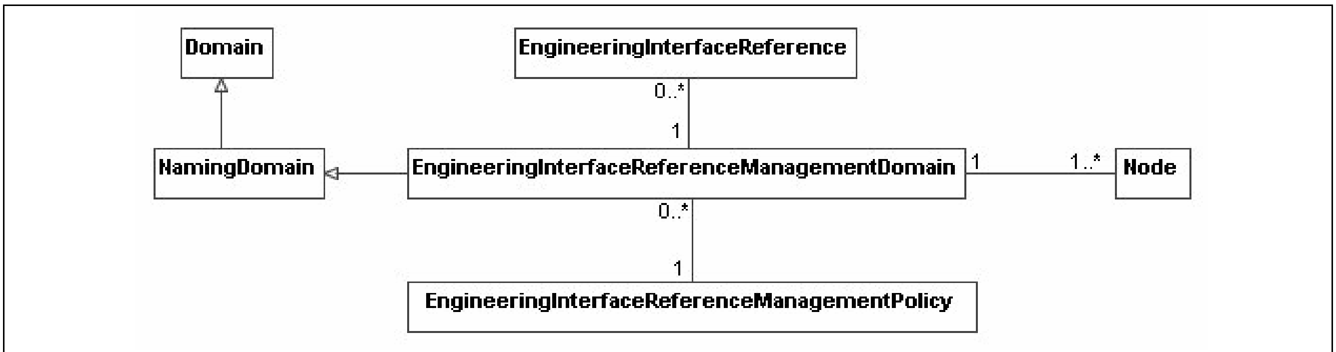



**Figure 25 – Engineering language model – Identifiers**

#### 10.1.5.7 Checkpoint

This part is about *checkpoints* and *checkpointings*.

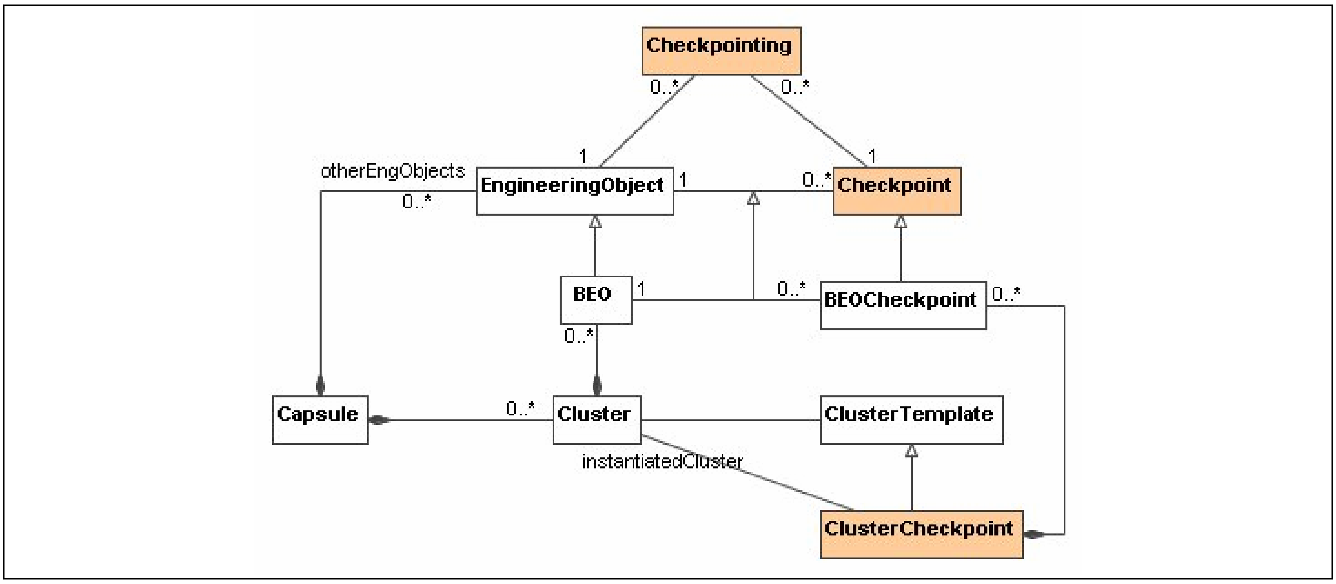



**Figure 26 – Engineering language model – Checkpoints**

#### 10.1.5.8 Other functions

The following diagrams show the concepts related to the *deactivation*, *cloning*, *recovery*, *reactivation* and *migration* functions.

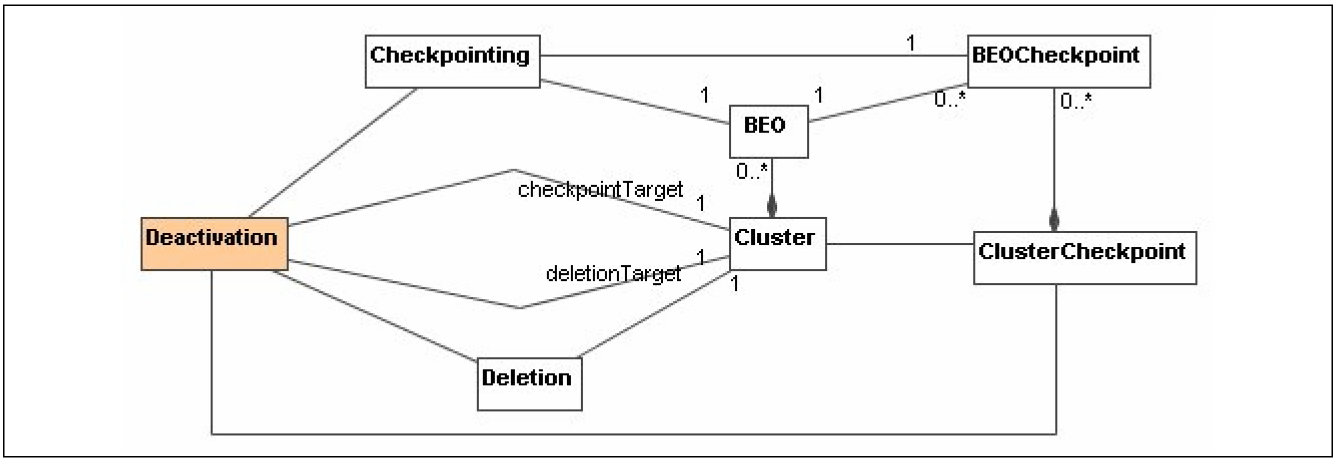



**Figure 27 – Engineering language model – Deactivation function**

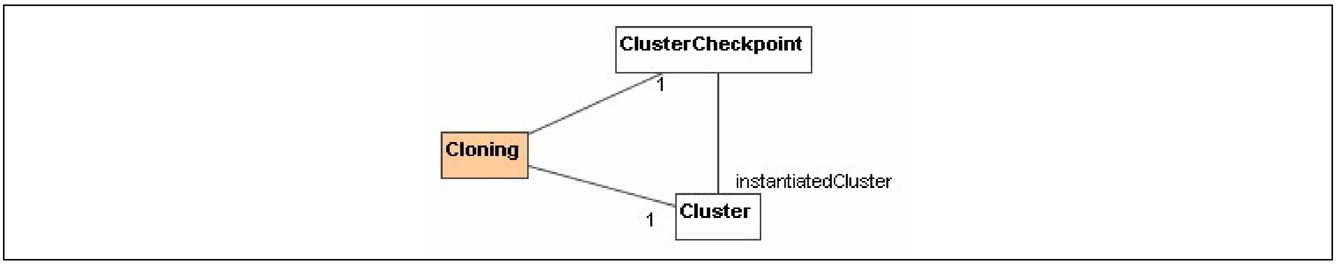



**Figure 28 – Engineering language model – Cloning function**

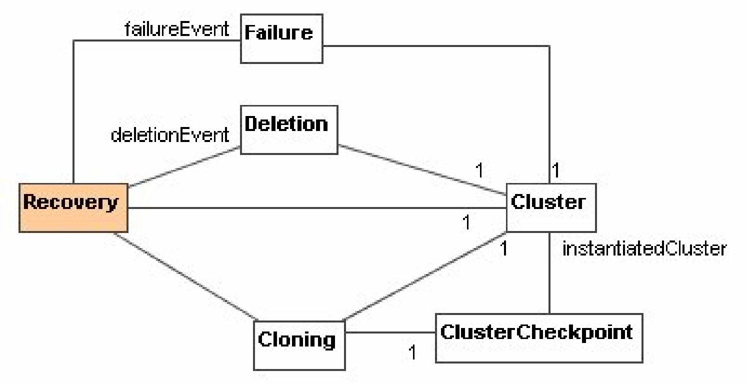



**Figure 29 – Engineering language model – Recovery function**

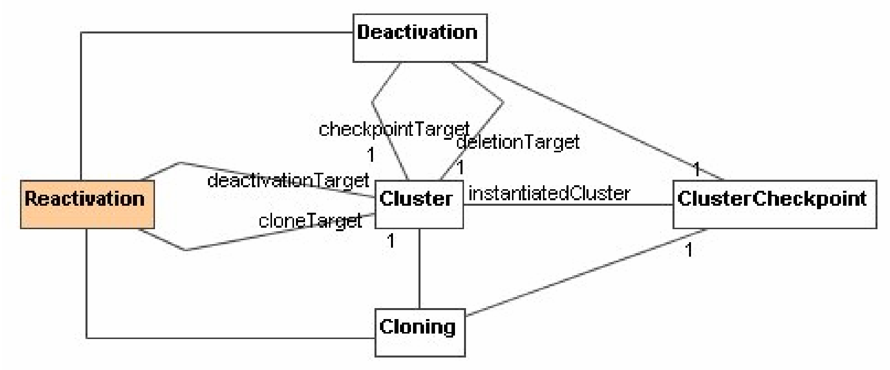



**Figure 30 – Engineering language model – Reactivation function**

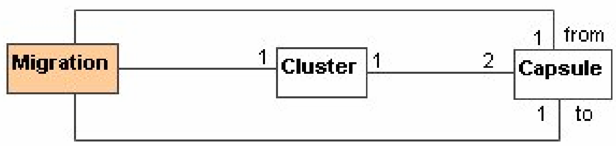



**Figure 31 – Engineering language model – Migration function**

## 10.2 UML mappings

The following paragraphs describe how the ODP engineering concepts described in the previous Clause are represented in UML in a computational specification. A brief explanation of the UML concepts used in the representation of each concept is given, together with a justification of the representation used.

NOTE – In this clause mappings are only defined for those concepts for which use has been demonstrated through an example, included in the main body of this document or in its annexes. Where no example has been identified, the concept concern is mentioned, but no mapping is offered.

### 10.2.1 Engineering object template

An *engineering object template* is modelled as a UML component. A UML component represents a modular part of a system which encapsulates its contents, and defines its behaviour in terms of provided and required interfaces through its ports.

The attribute isIndirectlyInstantiated of the component stereotyped «NV_ObjectTemplate» should be set to false. This attribute constraints the he kind of instantiation that applies to a UML component. If false, the component is instantiated as an addressable object.

The stereotype has the following attributes:

- **deployedNode**: **String** (defines a reference to a node where an engineering object is deployed).

- **securityDomain**: **String** (defines a reference of a security domain it may belong to).
- **managementDomain**: **String** (defines a reference of a management domain it may belong to).

#### 10.2.1 Engineering object

An *engineering object* is modelled as a UML InstanceSpecification of component, stereotyped as «NV_Object», since it is an instantiation of an engineering object template. An InstanceSpecification of component is an instance of a UML classifier component.

Basic engineering objects are particular kinds of engineering objects. Therefore, stereotype «NV_BEO» that identify such objects, inherits from «NV_Object»

#### 10.2.2 Cluster

A *cluster* is modelled as a UML InstanceSpecification of component, stereotyped as «NV_Cluster». This includes a configuration of *basic engineering objects* and has *bindings* to required *channels* for *communication*.

#### 10.2.3 Cluster manager

A *cluster manager* is modelled as a UML InstanceSpecification of component, stereotyped as «NV_ClusterManager».

#### 10.2.4 Capsule

A *capsule* is modelled as a UML InstanceSpecification of component, stereotyped as «NV_Capsule».

#### 10.2.5 Capsule manager

A *capsule manager* is modelled as a UML InstanceSpecification of component, stereotyped as «NV_CapsuleManager».

#### 10.2.6 Nucleus

A *nucleus* is modelled as a UML InstanceSpecification of component, stereotyped as «NV_Nucleus».

#### 10.2.7 Node

A *node* is modelled as a UML InstanceSpecification of component, stereotyped as «NV_Node».

#### 10.2.8 Channel

A *channel* is modelled as a UML package, stereotyped as «NV_Channel». It consists of *stubs*, *binders*, *protocol objects*, and possibly *<X> interceptors*. It is also modelled with tag definition of Channel ID for a set of engineering objects (*stub*, *binder*, *protocol object* and *<X> interceptor*) comprising a *channel*.

#### 10.2.9 Stub

A *stub* is modelled as a UML InstanceSpecification of component, stereotyped as «NV_Stub».

#### 10.2.10 Binder

A *binder* is modelled as a UML InstanceSpecification of component, stereotyped as «NV_Binder».

#### 10.2.11 <X> Interceptor

An *interceptor* is modelled as a UML InstanceSpecification of component, stereotyped as «NV_Interceptor».

#### 10.2.12 Protocol object

A *protocol object* is modelled as a UML InstanceSpecification of component, stereotyped as «NV_ProtocolObject».

#### 10.2.13 Communication domain

A *communication domain* is modelled as a UML package, stereotyped as «NV_CommunicationDomain».

#### 10.2.14 Communication interface

A *communication interface* is modelled as a UML Port through which protocol object is associated with other protocol objects or interceptor for *communication*.

#### 10.2.15 Binding endpoint identifier

A *binding endpoint identifier* is modelled as a UML ValueSpecification.

### 10.2.16 Engineering interface reference

An *engineering interface reference* is modelled as a UML ValueSpecification.

### 10.2.17 Engineering interface reference management domain

An *engineering interface reference management domain* is modelled as a UML package, stereotyped as «NV_InterfaceReferenceManagementDomain».

### 10.2.18 Engineering interface reference management policy

An *engineering interface reference management policy* is modelled as a UML constraint, stereotyped as «NV_InterfaceReferenceManagementPolicy».

### 10.2.19 Cluster template

A *cluster template* is modelled as a UML component, stereotyped as «NV_ClusterTemplate». The InstanceSpecification of component represents initial states of the *cluster*.

### 10.2.20 Checkpoint

A *checkpoint* is modelled as a UML InstanceSpecification of component, stereotyped as «NV_Checkpoint». The InstanceSpecification of component represents checkpointed object's states at the time of checkpointing.

### 10.2.21 Checkpointing

A *checkpointing* is modelled as a UML activity, UML interface, and UML action stereotyped as «NV_Checkpointing».

### 10.2.22 Cluster checkpoint

A *cluster checkpoint* is modelled as a UML InstanceSpecification of component, stereotyped as «NV_ClusterCheckpoint». The InstanceSpecification of component represents checkpointed cluster's state at the time of checkpointing.

### 10.2.23 Deactivation

A *deactivation* is modelled as a UML activity, UML interface, or an UML action stereotyped as «NV_Deactivation».

### 10.2.24 Cloning

A *cloning* is modelled as a UML activity, UML interface, or an UML action stereotyped as «NV_Cloning».

### 10.2.25 Recovery

A *recovery* is modelled as a UML activity, UML interface, or an UML action stereotyped as «NV_Recovery».

### 10.2.26 Reactivation

A *reactivation* is modelled as a UML activity, UML interface, or an UML action stereotyped as «NV_Reactivation».

### 10.2.27 Migration

A *migration* is modelled as a UML activity, UML interface, or an UML action stereotyped as «NV_Migration».

### 10.2.30 Summary of the UML mappings for the engineering language

The Engineering language profile (NV_Profile) specifies how the engineering viewpoint modelling concepts relate to and are represented in standard UML using stereotypes, tag definitions, and constraints. It represents the concepts of the engineering language model (see [10.1.28]).

The following shows diagrammatic representations of this UML profile.

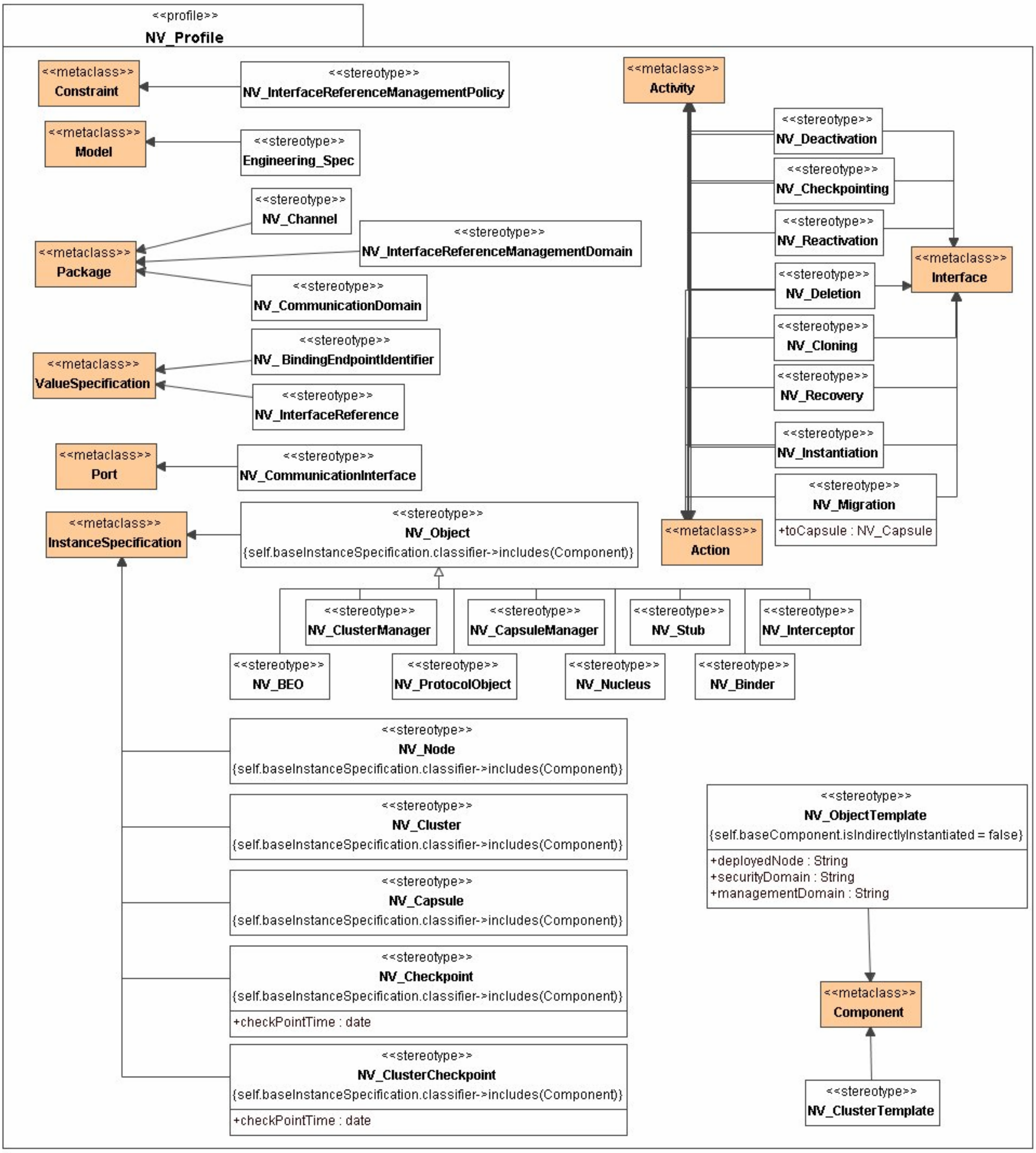



**Figure 32 – Graphical representation of the Engineering Language profile (using the UML 2.0 notation)**

NOTE 1 – From the diagrams above, infrastructure mechanisms are not well represented with the use of modelling language. It may be necessary to introduce functional objects, like the one in ODP Trader, such as recovery manager, etc. to cover above and ODP functions as well.

NOTE 2 – Not all management functions are shown in the above figure, e.g. thread management for Nucleus.

## 10.3 Engineering specification structure (in UML terms)

An engineering specification defines the infrastructure required to support functional distribution of an ODP system, by

- identifying the ODP *functions* required to manage physical distribution, communication, processing and storage;

- identifying the *roles* of different *engineering objects* supporting the ODP functions (for example the *nucleus*).

  Temporary note – How can we refer ODP functions from within our Engineering UML model? And which part (e.g. engineering part only) of ODP functions should we refer. E.g. Trading function standard has enterprise, information, and computational specifications within it.

An engineering specification is expressed in terms of

- a configuration of *engineering objects*, structured as *clusters*, *capsule* and *nodes* (that will be expressed with UML component diagrams, including InstanceSpecification of Component for *capsule*, *clusters* basic *engineering objects*, *capsule manager*, *cluster manager*, and *nucleus*);
- the *activities* that occur within those *engineering objects* (that will be expressed with UML Activity diagrams);
- the *interactions* of those *engineering objects* (that will be expressed with UML Sequence diagrams).

An engineering specification is constrained by the rules of the *engineering language*. These comprise

- channel rules [Part 3 – 8.2.1], interface reference rules [Part 3 – 8.2.2], distributed binding rules [Part 3 – 8.2.3] and relocation rules [Part 3 – 8.2.4] for the provision of distribution transparent interaction among *engineering objects*;
- cluster rules [Part 3 – 8.2.5], capsule rules [Part 3 – 8.2.6] and node rules [Part 3 – 8.2.7] governing the *configuration* of *engineering objects*;
- failure rules [Part 3 – 8.2.9].

Those rules will be expressed with UML or OCL constraints for relevant UML model elements.

All the UML elements representing the Engineering specification will be defined within a UML model, stereotyped «Engineering_Spec». Such a model contains UML packages that represent:

- structure of a *node*, including *nucleus*, *capsules*, *capsule managers*, *clusters*, *cluster managers*, *stub*, *binder*, *protocol objects*, *interceptors*, and *basic engineering objects*, with UML component diagram,
- *channels*, with UML component diagram and package,
- *domains*, with UML package
- *interactions* among those *engineering objects*, with UML activity diagrams, state charts and interaction diagrams.

## 10.4 Viewpoint correspondences for the engineering viewpoint specifications

### 10.4.1 Engineering and computational viewpoint specification correspondences

NOTE – The correspondence between engineering viewpoint specification and computational viewpoint specification can be derived from [9.4.4]

### 10.4.2 Engineering and technology viewpoint specification correspondences

Each *engineering object* corresponds to a set of one or more *technology objects*. The correspondence and implementable standards for each *technology object* is dependent on the choice of technology.

The engineering viewpoint specification does not have any correspondence to implementation.

*Engineering objects* and their *interfaces* correspond to *technology objects* and their *interfaces*, and thus will become *basic information source for testing* in the technology viewpoint.

# 11 Technology Specification

## 11.1 Modelling concepts

The modelling concepts used in a technology specification are defined, together with the structuring rules for their use, in Clause 9 of Part 3 of RM-ODP. The explanations of the concepts in the text that follows are not normative, and in case of conflicts between these explanations and the text of Part 3, the latter should be followed.

### 11.1.1 Implementable standard

A template for a *technology object*.

#### 11.1.2 Implementation

A process of instantiation whose validity can be subject to test.

#### 11.1.3 IXIT

*Implementation* eXtra Information for Test.

#### 11.1.4 Model of the technology language

The diagram below illustrates the concepts of the technology language and the relationships between them.

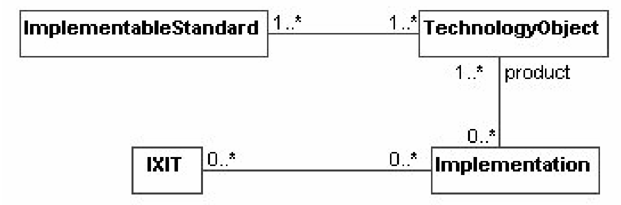



**Figure 33 – Model of the technology language**

### 11.2 UML mappings

The following paragraphs describe how the ODP technology concepts described in the previous clause are represented in UML in a technology specification. A brief explanation of the UML concepts used in the representation of each concept is given, together with a justification of the representation used.

NOTE – In this clause mappings are only defined for those concepts for which use has been demonstrated through an example, included in the main body of this document or in its annexes. Where no example has been identified, the concept concern is mentioned, but no mapping is offered.

#### 11.2.1 Technology object

A *technology object* is modelled as a UML InstanceSpecification of artifact or node, stereotyped as «TV_Object». A UML InstanceSpecification of artifact represents implementation or realization of functionality identified in its engineering viewpoint specification. A UML InstanceSpecification of node represents a run-time computational resource, such as computer, including execution environment for deployed artifacts.

#### 11.2.2 Technology object type

*Technology object types* can be used to characterize the different kinds of *technology objects* that are used in a technology specification (such as PCs, application servers, LANs, WANs, etc.). A *technology object type* is modelled as UML artifact or node, stereotyped as «TV_ObjectType». They will act as valid classifiers for the UML InstanceSpecifications of artifact or node, stereotyped as «TV_Object», that model the corresponding *technology objects* that conform to such types.

#### 11.2.3 Implementable standard

An *implementation standard* is modelled as a UML component, stereotyped as «TV_ImplementationStandard».

#### 11.2.4 Implementation

An *implementation* is modelled as a UML activity, stereotyped as «TV_Implementation».

#### 11.2.5 IXIT

An *IXIT* is modelled as a UML comment, stereotyped as «TV_IXIT».

#### 11.2.6 Summary of the UML mappings for the technology language

The Technology language profile (TV_Profile) specifies how the engineering viewpoint modelling concepts relate to and are represented in standard UML using stereotypes, tag definitions, and constraints. It represents the concepts of the technology language model (see [11.1.4]).

The following shows diagrammatic representations of this UML profile. See Clause [A.5] for a detailed specification of the stereotypes described here.

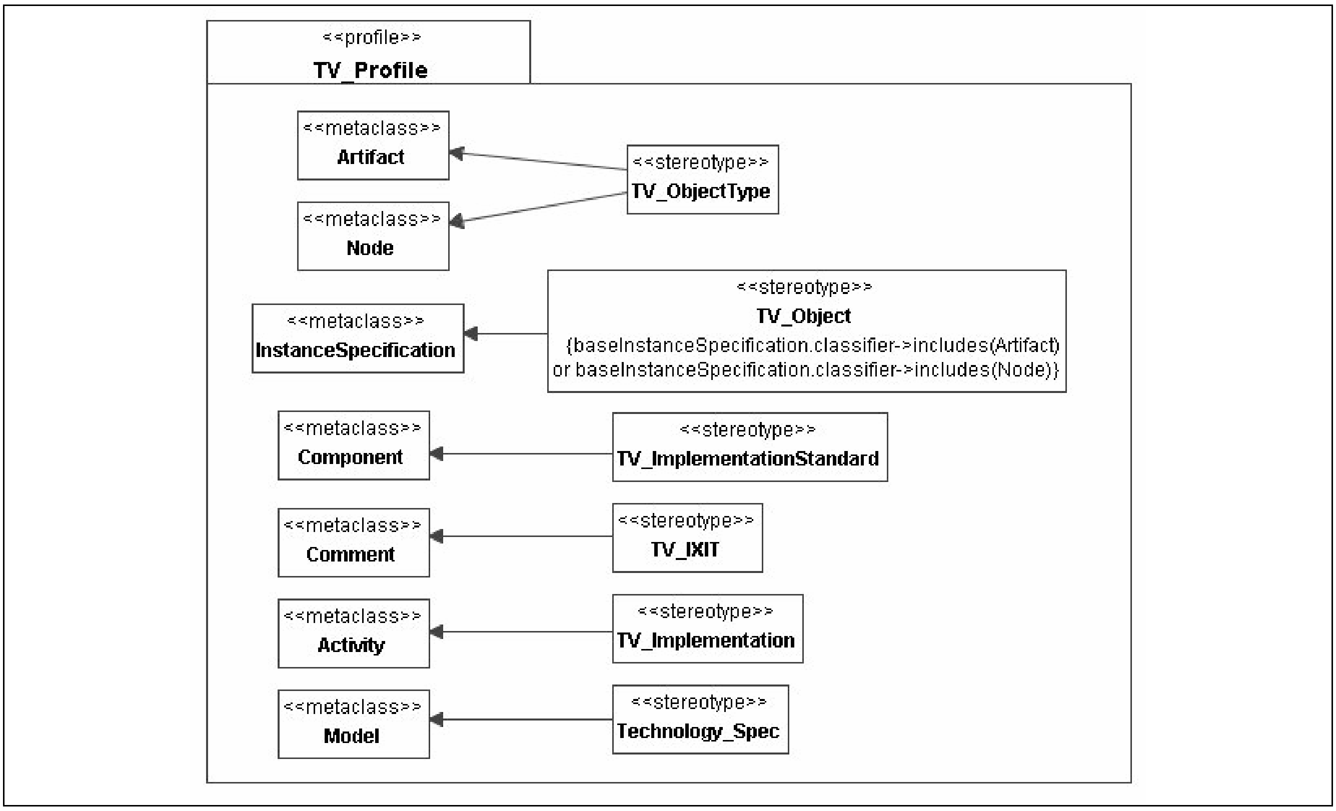



**Figure 34 – Graphical representation of the Technology Language profile (using the UML 2.0 notation)**

The following restrictions apply to the elements depicted in Figure 34. They are derived from the corresponding constraints on the elements shown in Figure 33 and on their relationships:

- Every t*echnology object type* is associated with at least one *implementable standard*.
- Every *implementation standard* is associated with (or is implemented as) one or more *technology objects*.
- Every *implementation* is associated with (or produces) one or more *technology objects*.

## 11.3 Technology specification structure (in UML terms)

A technology specification defines the choice of technology for an ODP system in terms of

- a configuration of *technology objects*, and
- *interfaces* between the *technology objects*.

NOTE 1 – Links between deployment boxes may be used to represent physical communication lines (e.g. to express multiple lines for redundancies).

NOTE 2 – Network (e.g. the Internet) may be expressed with a deployment box connected with other deployment boxes.

A technology specification states:

- How the specifications for an ODP system are implemented, which may be expressed with component instances and the relationships between them with text explanation.
- Taxonomy of such specifications, which may be provided with name(s) of *implementable standards* described in stereotyped notes attached to deployment diagram including component instance diagram.
- Information required from implementers to support testing, which may be specified with stereotyped note describing IXIT.

NOTE – Software architecture styles, such as SOA, MVC and N-tier, are considered mainly in the engineering viewpoint, since they are closely related to distribution strategy. See Annex D.

All the UML elements representing the technology specification will be defined within a UML model, stereotyped «Technology_Spec». Such a model contains UML packages that represent:

- structure of a *node* instance, including *node* instances within a *node* instance, artifacts, and networks, with deployment diagram, and
- communication links among *nodes*, with deployment diagram.

### 11.4 Viewpoint correspondences for the technology viewpoint

A set of one or more *technology objects* correspond to an *engineering object*, and they implement specified functionality in corresponding *engineering object* in technology specific way.

Note that a choice of specific technology in technology viewpoint may have constraint effect on the possible architecture/platform styles/patterns and deployment patterns in engineering viewpoint specifications.

Temporary Note – In a WG19-J meeting, several points were made as possible requirements on Technology viewpoint.

- One participant made the following statements. Within a large-scale procurement process, it is often the case that Technology viewpoint specification (i.e. hardware, network, operating systems, middleware, database etc.) comes first, or comes earlier than Computational or Engineering viewpoint specification. In these circumstances, this early Technology viewpoint specification plays a role of constraints to the choice of architecture for Computational and Engineering viewpoint specifications. This possibility (reverse-direction influencing) may be noted somewhere in the standards.
- A capability of specifying 1) number of instances (e.g. number of client systems like 10 thousand clients communicating with 2 web servers), and 2) location of instances (e.g. one server in Tokyo and the other in Geneva) should also be a candidate target for UML for technology viewpoint. If we were to specify or describe the mapping of non-functional requirements (e.g. performance) on Computational, Engineering, and Technology viewpoint specifications, the capability of specifying number of instances may be an important aspect of the mapping architecture and mapping specification.

## 12 Correspondences specification

Temporary Note – Following discussion in Bari, and in the WODPEC 2005 workshop, the WG concluded at Bari that a further profile is required, which addresses correspondences between model elements in different viewpoints. NBs are requested to provide contributions on how ODP correspondences can be mapped to UML 2.0.

## 13 Conformance and compliance

### 13.1 Conformance

Levels of conformance may vary. At the least, implementations of tools claiming conformance to this document must support:

- one or more of the UML profiles for viewpoint languages defined in Clauses 7 to 11; further conformance may be claimed if the tool concerned supports policing or enforcing of the constraints specified for the stereotypes defined in the relative profiles.
- tracing mechanisms to enable specification of the correspondences between ODP modelling elements in the various viewpoint models supported by the tool, as defined in Clauses 7.4, 8.4, 9.4, 10.4, and 11.4;
- the structuring style for ODP system specifications defined in Clause 6.5.

### 13.2 Compliance

Specifications claiming compliance with this document shall:

- use the structuring style defined in Clause 6.5:
- be expressed using the UML profiles for the viewpoint languages defined in Clauses 7 to 11 of this document;
- specify the correspondences between ODP modelling elements in different viewpoint models using the tracing mechanisms defined in Clause s 7.4, 8.4, 9.4, 10.4, and 11.4.

Temporary Note – New standards/specifications may deploy other concepts not defined here as profile elements such as stereotypes. Some concepts are not treated as extensions of UML elements (e.g. "scope" in enterprise language), and also Part 2 and Part 3 concepts may be used in the standards/specifications. NBs are requested to comment on whether it might be better to extend the coverage of this clause to include such issues.

# Annex A Summary of UML profiles of ODP languages using ITU-T guidelines for UML profile design

(This annex forms an integral part of this Recommendation | International Standard)

This annex contains the description of the UML Profiles for the five ODP viewpoint languages, written according to the ITU-T guidelines for UML Profile Design (ITU T Rec. Z.119, Specification and Description Language (SDL). Guidelines for UML Profile Design).

This annex is normative.

## A.1 Enterprise viewpoint

### A.1.1 ODP System

#### A.1.1.1 «EV_ODPSystem»

The stereotype «EV_ODPSystem» extends the metaclass class with multiplicity [0..1]. It is intended to capture the semantics of an *ODP System* in the RM-ODP enterprise language.

#### A.1.1.2 Attributes

No tag definitions are defined for this stereotype

#### A.1.1.3 Constraints

May only be applied to a class also stereotyped as «EV_EnterpriseObject».

#### A.1.1.4 Semantics

See [7.1.1] and [7.2.1].

#### A.1.1.5 Notation

UML standard syntax for UMLODP System with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| «EV_ODPSystem» |  |

#### A.1.1.6 References

RM-ODP: *ODP System* [Part 2 – 3.2.4]

UML: class [UML – 7.3.7]

### A.1.2 Field of Application

#### A.1.2.1 «FieldOfApplication»

The stereotype «FieldOfApplication» extends the metaclass Comment with multiplicity [0..1]. It is intended to capture the semantics of a *Field of Application* in the RM-ODP enterprise language.

#### A.1.2.2 Attributes

No tag definitions are defined for this stereotype

#### A.1.2.3 Constraints

May only be applied to a UML model stereotyped as «EnterpriseSpec».

#### A.1.2.4 Semantics

See [7.1.1] and [7.2.3].

#### A.1.2.5 Notation

UML standard syntax for Comment with stereotype is used.

#### A.1.2.6 References

RM-ODP: *Field of Application* [E/L – 6.1.2]

UML: Comment [UML – 7.3.9]

### A.1.3 Community

#### A.1.3.1 «EV_Community»

The stereotype «EV_Community» extends the metaclass Component with multiplicity [0..1]. It is intended to capture the semantics of a *Community* in the RM-ODP enterprise language.

#### A.1.3.2 Attributes

No tag definitions are defined for this stereotype.

#### A.1.3.3 Constraints

None specified.

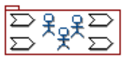
#### A.1.3.3 Semantics

See [7.1.2] and [7.2.4].

#### A.1.3.5 Notation

UML standard syntax for Component with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| «EV_Community» |  |

#### A.1.3.6 References

RM-ODP: *Community* [Part 3 – 5.1.1], [E/L – 7.1]

UML: Component [UML – 8.3.1]

### A.1.4 Community behaviour

#### A.1.4.1 «EV_CommunityBehaviour»

The stereotype «EV_CommunityBehaviour» extends the metaclass Realization with multiplicity [0..1]. It is intended to capture the semantics of a relatiohship between a *community* and the *behaviour* defined in its specification in the RM-ODP enterprise language.

#### A.1.4.2 Attributes

No tag definitions are defined for this stereotype.

#### A.1.4.3 Constraints

May only exist between a component, stereotyped as «EV_Community».

#### A.1.4.4 Semantics

See [7.1.2] and [7.2.4]

#### A.1.4.5 Notation

UML standard syntax for Realization with stereotype is used.

#### A.1.4.6 References

RM-ODP: *Community* [Part 3 – 5.1.1], [E/L – 7.1]

UML: Realization [UML – 8.3.4]

### A.1.5 Enterprise Object

#### A.1.5.1 «EV_EnterpriseObject»

The stereotype «EV_EnterpriseObject» extends the metaclass Class with multiplicity [0..1]. It is intended to capture the semantics of an *Enterprise Object* in the RM-ODP enterprise language.

#### A.1.5.2 Attributes

No tag definitions are defined for this stereotype

#### A.1.5.3 Constraints

None.

#### A.1.5.4 Semantics

See [7.1.2] and [7.2.5].

#### A.1.5.5 Notation

UML standard syntax for Class with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| «EV_EnterpriseObject» |  |

#### A.1.5.6 References

RM-ODP: *Object* [Part 2 – 8.1], *Enterprise Object* [E/L – 7.4]

UML: Class [UML – 7.3.7]

### A.1.6 Community Object

#### A.1.6.1 «EV_CommunityObject»

The stereotype «EV_CommunityObject» extends the metaclass Class with multiplicity [0..1]. It is intended to capture the semantics of a *Community Object* in the RM-ODP enterprise language.

#### A.1.6.2 Attributes

No tag definitions are defined for this stereotype

#### A.1.6.3 Constraints

1. May only be applied to a class also stereotyped as «EV_EnterpriseObject».

2. Within a well-formed complete model there must be a component, stereotyped as «EV_Community», expressing the refinement of the *Community Object*.

#### A.1.6.4 Semantics

See [7.1.2] and [7.2.6].

#### A.1.6.5 Notation

UML standard syntax for Class with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| «EV_CommunityObject» |  |

#### A.1.6.6 References

RM-ODP: *Object* [Part 2 – 8.1], *Community Object* [E/L – 6.2.2]

UML: Class [UML – 7.3.7]

### A.1.7 Refines as Community

#### A.1.7.1 «EV_RefinesAsCommunity»

The stereotype «EV_RefinesAsCommunity» extends the metaclass Dependency with multiplicity [0..1]. It is intended to capture the semantics of refinement of a *Community Object* as a *Community* in the RM-ODP enterprise language.

#### A.1.7.2 Attributes

No tag definitions are defined for this stereotype

#### A.1.7.3 Constraints

May only exist from a class, stereotyped as «EV_CommunityObject», to a component, stereotyped as «EV_Community»

#### A.1.7.4 Semantics

See [7.1.2] and [7.2.6].

#### A.1.7.5 Notation

UML standard syntax for UML dependency with stereotype is used.

#### A.1.7.6 References

RM-ODP: *Object* [Part 2 – 8.1], *Community Object* [E/L – 6.2.2]

UML: Dependency [UML – 7.3.12]

### A.1.8 Objective

#### A.1.8.1 «EV_Objective»

The stereotype «EV_Objective» extends the metaclass Class with multiplicity [0..1]. It is intended to capture the semantics of an *Objective* in the RM-ODP enterprise language.

#### A.1.8.2 Attributes

No tag definitions are defined for this stereotype

#### A.1.8.3 Constraints

None

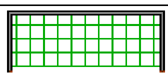
#### A.1.8.4 Semantics

See [7.1.2] and [7.2.7]

#### A.1.8.5 Notation

UML standard syntax for Class with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| «EV_Objective» |  |

#### A.1.8.6 References

RM-ODP: *Objective* [E/L – 6.2.1]

UML: Class [UML – 7.3.7]

### A.1.9 Objective (of a Community)

#### A.1.9.1 «EV_ObjectiveOf»

The stereotype «EV_ObjectiveOf» extends the metaclass Association with multiplicity [0..1]. It is intended to capture the semantics of an *Objective* (of a *Community*) in the RM-ODP enterprise language.

#### A.1.9.2 Attributes

No tag definitions are defined for this stereotype

#### A.1.9.3 Constraints

May only occur between a class stereotyped as «EV_Objective» and a component stereotyped as «EV_Community»

#### A.1.9.4 Semantics

See [7.1.2] and [7.2.7]

#### A.1.9.5 Notation

UML standard syntax for Association with stereotype is used.

#### A.1.9.6 References

RM-ODP: *Objective* [E/L – 6.2.1]

UML: Association [UML – 7.3.3]

### A.1.10 Contract

#### A.1.10.1 «EV_CommunityContract»

The stereotype «EV_CommunityContract» extends the metaclass Package with multiplicity [0..1]. It is intended to capture the semantics of a *Contract* in the RM-ODP enterprise language.

#### A.1.10.2 Attributes

No tag definitions are defined for this stereotype

#### A.1.10.3 Constraints

In a well-formed complete model there must be a component, stereotyped as «EV_Community», expressing the specification of the *Community* within the namespace of the package expressing the *Community Contract*.

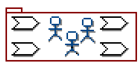
#### A.1.10.4 Semantics

See [7.1.2] and [7.2.8].

#### A.1.10.5 Notation

UML standard syntax for Package with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| «EV_CommunityContract» |  |

#### A.1.10.6 References

RM-ODP: *contract* [Part 2 – 11.2.1], [E/L – 7.3]

UML: Package [UML – 7.3.37]

### A.1.11 Role

#### A.1.11.1 «EV_Role»

The stereotype «EV_Role» extends the metaclass Class and ActivityPartition with multiplicity [0..1]. It is intended to capture the semantics of a *Role* in the RM-ODP enterprise language.

#### A.1.11.2 Attributes

No tag definitions are defined for this stereotype

#### A.1.11.3 Constraints

None

#### A.1.11.4 Semantics

See [7.1.2], [7.2.9.2] and [7.2.9.3]

#### A.1.11.5 Notation

UML standard syntax for Class or ActivityPartition with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| «EV_Role» |  |

#### A.1.11.6 References

RM-ODP: *Role* [Part 2 – 9.14]

UML: Class [UML – 7.3.7]; ActivityPartition [UML – 12.3.10].

### A.1.12 Role fulfilment

#### A.1.12.1 «EV_FulfilsRole»

The stereotype «EV_FulfilsRole» extends the metaclass Association with multiplicity [0..1]. It is intended to capture the semantics of a *Role fulfilment* in the RM-ODP enterprise language.

#### A.1.12.2 Attributes

No tag definitions are defined for this stereotype.

#### A.1.12.3 Constraints

May only be between a class, stereotyped as «EV_EnterpriseObject» and a class, stereotyped as «EV_Role».

#### A.1.12.4 Semantics

See [7.1.2] and [7.2.5].

#### A.1.12.5 Notation

UML standard syntax for Association with stereotype is used.

#### A.1.12.6 References

RM-ODP: *Role* [Part 2 – 9.14]

UML: Association [UML – 7.3.3]

### A.1.13 Interaction

#### A.1.13.1 «EV_Interaction»

The stereotype «EV_Interaction» extends the metaclass Class with multiplicity [0..1]. It is intended to capture the semantics of an *interaction* in the RM-ODP enterprise language.

#### A.1.13.2 Attributes

No tag definitions are defined for this stereotype

#### A.1.13.3 Constraints

None.

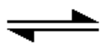
#### A.1.13.4 Semantics

See [7.1.2] and [7.2.9.2].

#### A.1.13.5 Notation

UML standard syntax for Class with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| «EV_Interaction» |  |

#### A.1.13.6 References

RM-ODP: *Interaction* [Part 2 – 8.3]

UML: Class [UML – 7.3.7]

### A.1.14 Interaction Initiator

#### A.1.14.1 «EV_InteractionInitiator»

The stereotype «EV_InteractionInitiator» extends the metaclass Association with multiplicity [0..1]. It is intended to capture the semantics of a *Interaction Initiator* in the RM-ODP enterprise language.

#### A.1.14.2 Attributes

No tag definitions are defined for this stereotype

#### A.1.14.3 Constraints

May only occur between a class stereotyped as «EV_Interation» and a class stereotyped as «EV_Role»

#### A.1.14.4 Semantics

See [7.2.9.2]

#### A.1.14.5 Notation

UML standard syntax for Association with stereotype is used.

#### A.1.14.6 References

RM-ODP: *Initiating object* [Part 2 – 13.3.1]

UML: Association [UML – 7.3.3]

### A.1.15 Interaction Responder

#### A.1.15.1 «EV_InteractionResponder»

The stereotype «EV_InteractionResponder» extends the metaclass Association with multiplicity [0..1]. It is intended to capture the semantics of a *Interaction Responder* in the RM-ODP enterprise language.

#### A.1.15.2 Attributes

No tag definitions are defined for this stereotype.

#### A.1.15.3 Constraints

May only occur between a class stereotyped as «EV_Interation» and a class stereotyped as «EV_Role»

#### A.1.15.4 Semantics

See [7.2.9.2].

#### A.1.15.5 Notation

UML standard syntax for Association with stereotype is used.

#### A.1.15.6 References

RM-ODP: *Responding object* [Part 2 – 13.3.1]

UML: Association [UML – 7.3.3]

### A.1.16 Artefact

#### A.1.16.1 «EV_Artefact»

The stereotype «EV_Artefact» extends the metaclasses Signal and ObjectNode with multiplicity [0..1]. It is intended to capture the semantics of a *artefact* in the RM-ODP enterprise language.

#### A.1.16.2 Attributes

No tag definitions are defined for this stereotype.

#### A.1.16.3 Constraints

None.

#### A.1.16.4 Semantics

See [7.1.2], [7.2.9.2] (use with Signal) and [7.2.9.3] (use with ObjectNode).

#### A.1.16.5 Notation

UML standard syntax for Signal and ObjectNode with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| «EV_Artefact» |  |

#### A.1.16.6 References

RM-ODP: *Object* [Part 2 – 8.1], *Artefact* [E/L – 6.3.2]

UML: Signal [UML – 13.2.23]; ObjectNode [UML – 13.3.38].

### A.1.17 Artefact role

#### A.1.17.1 «EV_ArtefactRole»

The stereotype «EV_ArtefactRole» extends the metaclass Association with multiplicity [0..1]. It is intended to capture the semantics of an *artefact role* of an *enterprise object* when it is referenced in an *action*, in the RM-ODP enterprise language.

#### A.1.17.2 Attributes

No tag definitions are defined for this stereotype.

#### A.1.17.3 Constraints

May only be between a class, stereotyped as «EV_EnterpriseObject» and a signal, stereotyped as «EV_Artefact».

#### A.1.17.4 Semantics

See [7.1.2] and [7.2.11].

#### A.1.17.5 Notation

UML standard syntax for Association with stereotype is used.

#### A.1.17.6 References

RM-ODP: *Object* [Part 2 – 8.1], *Artefact* [E/L – 6.3.2]

UML: Association [UML – 7.3.3]

### A.1.18 Artefact reference

#### A.1.18.1 «EV_ArtefactReference»

The stereotype «EV_ArtefactReference» extends the metaclass Association with multiplicity [0..1]. It is intended to capture the semantics of a reference to an *artefact* in an *action*, in the RM-ODP enterprise language.

#### A.1.18.2 Attributes

No tag definitions are defined for this stereotype

#### A.1.18.3 Constraints

May only be between a class, stereotyped as «EV_Interaction» and a signal, stereotyped as «EV_Artefact».

#### A.1.18.4 Semantics

See [7.1.2] and [7.2.11].

#### A.1.18.5 Notation

UML standard syntax for Association with stereotype is used.

#### A.1.18.6 References

RM-ODP: *Object* [Part 2 – 8.1], *Artefact* [E/L – 6.3.2].

UML: Association [UML – 7.3.3].

### A.1.19 Process

#### A.1.19.1 «EV_Process»

The stereotype «EV_Process» extends the metaclass Activity with multiplicity [0..1]. It is intended to capture the semantics of a *Process* in the RM-ODP enterprise language.

#### A.1.19.2 Attributes

No tag definitions are defined for this stereotype.

#### A.1.19.3 Constraints

None.

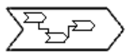
#### A.1.19.4 Semantics

See [7.1.2] and [7.2.9.3].

#### A.1.19.5 Notation

UML standard syntax for Activity with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| «EV_Process» |  |

#### A.1.19.6 References

RM-ODP: *Process* [E/L – 6.3.5]

UML: Activity [UML – 12.3.4]

### A.1.20 Step

#### A.1.20.1 «EV_Step»

The stereotype «EV_Step» extends the metaclass Action with multiplicity [0..1]. It is intended to capture the semantics of a *Step* in the RM-ODP enterprise language.

#### A.1.20.2 Attributes

No tag definitions are defined for this stereotype.

#### A.1.20.3 Constraints

None.

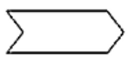
#### A.1.20.4 Semantics

See [7.1.2] and [7.2.9.3].

#### A.1.20.5 Notation

UML standard syntax for Action with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| «EV_Step» |  |

#### A.1.20.6 References

RM-ODP: *Action* [Part 2 – 8.3], *Step* [E/L – 6.3.6].

UML: Action [UML – 12.3.2].

### A.1.21 Policy envelope

#### A.1.21.1 «EV_PolicyEnvelope»

The stereotype «EV_PolicyEnvelope» extends the metaclass Class with multiplicity [0..1]. It is intended to capture the semantics of a *Policy envelope* in the RM-ODP enterprise language.

#### A.1.21.2 Attributes

No tag definitions are defined for this stereotype.

#### A.1.21.3 Constraints

None.

#### A.1.21.4 Semantics

See [7.1.3] and [7.2.13]

#### A.1.21.5 Notation

UML standard syntax for Class with stereotype is used.

#### A.1.21.6 References

RM-ODP: *Policy* [Part 2 – 11.2.4], [E/L – 6.4.1, 7.9].

UML: Class [UML – 7.3.7].

### A.1.22 Policy value

#### A.1.22.1 «EV_PolicyValue»

The stereotype «EV_PolicyValue» extends the metaclass Class with multiplicity [0..1]. It is intended to capture the semantics of a *Policy value* in the RM-ODP enterprise language.

#### A.1.22.2 Attributes

No tag definitions are defined for this stereotype.

#### A.1.22.3 Constraints

None.

#### A.1.22.4 Semantics

See [7.1.3] and [7.2.13].

#### A.1.22.5 Notation

UML standard syntax for Class with stereotype is used.

#### A.1.22.6 References

RM-ODP: *Policy* [Part 2 – 11.2.4], [E/L – 6.4.1, 7.9].

UML: Class [UML – 7.3.7].

### A.1.23 Controlling Authority

#### A.1.23.1 «EV_ControllingAuthority»

The stereotype «EV_ControllingAuthority» extends the metaclass Association with multiplicity [0..1]. It is intended to capture the semantics of a *Controlling Authority* in the RM-ODP enterprise language.

#### A.1.23.2 Attributes

No tag definitions are defined for this stereotype

#### A.1.23.3 Constraints

May only occur between a class stereotyped as «EV_PolicyValue» and a class stereotyped either as «EV_Process» or «EV_Interaction».

#### A.1.23.4 Semantics

See [7.1.3] and [7.2.13].

#### A.1.23.5 Notation

UML standard syntax for Association with stereotype is used.

#### A.1.23.6 References

RM-ODP: *Policy* [Part 2 – 11.2.4], [E/L – 6.4.1, 7.9].

UML: Association [UML – 7.3.3].

### A.1.24 Policy Envelope Rule

#### A.1.24.1 «EV_PolicyEnvelopeRule»

The stereotype «EV_PolicyEnvelopeRule» extends the metaclass Constraint with multiplicity [0..1]. It is intended to capture the semantics of a *Policy* envelope rule in the RM-ODP enterprise language.

#### A.1.24.2 Attributes

No tag definitions are defined for this stereotype.

#### A.1.24.3 Constraints

May only apply to a Class stereotyped as «EV_PolicyEnvelope».

#### A.1.24.4 Semantics

See [7.1.3] and [7.2.13].

#### A.1.24.5 Notation

UML standard syntax for Constraint with stereotype is used.

#### A.1.24.6 References

RM-ODP: *Policy* [Part 2 – 11.2.4], [E/L – 6.4.1, 7.9].

UML: Constraint [UML – 7.3.10].

### A.1.25 Policy Value Rule

#### A.1.25.1 «EV_PolicyValueRule»

The stereotype «EV_PolicyValueRule» extends the metaclass Constraint with multiplicity [0..1]. It is intended to capture the semantics of a *Policy* value rule in the RM-ODP enterprise language.

#### A.1.25.2 Attributes

No tag definitions are defined for this stereotype.

#### A.1.25.3 Constraints

May only apply to a Class stereotyped as «EV_PolicyValue».

#### A.1.25.4 Semantics

See [7.1.3] and [7.2.13].

#### A.1.25.5 Notation

UML standard syntax for Constraint with stereotype is used.

#### A.1.25.6 References

RM-ODP: *Policy* [Part 2 – 11.2.4], [E/L – 6.4.1, 7.9].

UML: Constraint [UML – 7.3.10].

### A.1.26 Affected behaviour

#### A.1.26.1 «EV_AffectedBehaviour»

The stereotype «EV_AffectedBehaviour» extends the metaclass Dependency with multiplicity [0..1]. It is intended to capture the semantics of constraint on *behaviour* by a *policy* in the RM-ODP enterprise language.

#### A.1.26.2 Attributes

No tag definitions are defined for this stereotype.

#### A.1.26.3 Constraints

May only exist from a class, stereotyped as «EV_Role» or «EV_Interaction», or an activity, stereotyped as «EVProcess» to a class, stereotyped as «EV_PolicyEnvelope».

#### A.1.26.4 Semantics

See [7.1.3] and [7.2.13].

#### A.1.26.5 Notation

UML standard syntax for Dependency with stereotype is used.

#### A.1.26.6 References

RM-ODP: *Policy* [Part 2 – 11.2.4], [E/L – 6.4.1, 7.9].

UML: Dependency [UML – 7.3.12].

### A.1.27 Party

#### A.1.27.1 EV_Party

The stereotype «EV_Party» extends the metaclass Class with multiplicity [0..1]. It is intended to capture the semantics of a *Party* in the RM-ODP enterprise language.

#### A.1.27.2 Attributes

No tag definitions are defined for this stereotype.

#### A.1.27.3 Constraints

May only be applied to a class that is also stereotyped as «EV_EnterpriseObject».

#### A.1.27.4 Semantics

See [7.1.4] and [7.2.20].

#### A.1.27.5 Notation

UML standard syntax for Class with stereotype is used.

#### A.1.27.6 References

RM-ODP: *Object* [Part 2 – 8.1], *Enterprise Object* [E/L – 7.4], *Party* [E/L – 6.5.1].

UML: Class [UML – 7.3.1].

### A.1.28 Accountable action

#### A.1.28.1 «EV_Accountable»

The stereotype «EV_Accountable» extends the metaclass Association with multiplicity [0..1]. It is intended to capture the semantics of an *Accountable action* in the RM-ODP enterprise language.

#### A.1.28.2 Attributes

No tag definitions are defined for this stereotype.

#### A.1.28.3 Constraints

May only exist between a class, stereotyped as «EV_Role», that has an association, stereotyped as «EV_FulfilsRole», with a class, stereotyped as «EV_Party», and either a class, stereotyped as «EV_Interaction» or an activity, stereotyped as «EV_Process».

#### A.1.28.4 Semantics

See [7.1.4] and [7.2.21].

#### A.1.28.5 Notation

UML standard syntax for Association with stereotype is used.

#### A.1.28.6 References

RM-ODP: *Accountability* concepts [E/L – 6.5].

UML: Association [UML – 7.3.3].

### A.1.29 Delegation

#### A.1.29.1 «EV_Delegation»

The stereotype «EV_Delegation» extends the metaclass Association with multiplicity [0..1]. It is intended to capture the semantics of a *Delegation* in the RM-ODP enterprise language.

#### A.1.29.2 Attributes

No tag definitions are defined for this stereotype.

#### A.1.29.3 Constraints

May only be applied to associations between a class, stereotyped as «EV_Role», which as an association, stereotyped as «EV_FulfilsRole», with a class stereotyped as «EV_Party», and a class stereotyped as «EV_Role».

#### A.1.29.4 Semantics

See [7.2.22].

#### A.1.29.5 Notation

UML standard syntax for Association with stereotype is used.

#### A.1.29.6 References

RM-ODP: *Delegation* [E/L – 6.5.4].

UML: Association [UML – 7.3.3].

## A.2 Information viewpoint

### A.2.1 Information object

#### A.2.1.1 «IV_Object»

The stereotype «IV_Object» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of an *information object* in RM-ODP information language.

#### A.2.1.2 Attributes

No tag definitions are defined for this stereotype.

#### A.2.1.3 Constraints

An *information object* is modelled as an instance of a class:

**context** IV_Object **inv:** self.baseInstanceSpecification.classifier->includes(Class)

#### A.2.1.4 Semantics

An *information object* is modelled as a UML object, which is an instance of a class, and therefore it is mapped to an «IV_Object» InstanceSpecification.

An InstanceSpecification is a UML model element that represents an instance in a modelled system. It specifies existence of an entity in a modelled system and completely or partially describes the entity. The description includes the classification of the entity by one or more classifiers of which the entity is an instance.

In UML, an object is an entity with a well-defined boundary and identity that encapsulates state and behaviour. State is represented by attributes and relationships. The behaviour of UML object mapping to ODP information objects is represented by state machines.

#### A.2.1.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| «IV_Object» |  |

#### A.2.1.6 References

RM-ODP: *Object* [Part 2 – 8.1], Information language [Part 3 – 6]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.2.2 Information object template

#### A.2.2.1 «IV_ObjectTemplate»

The stereotype «IV_ObjectTemplate» extends the metaclass Class with multiplicity [0..1]. It is intended to capture the semantics of *template* for *information objects* in the RM-ODP information language.

#### A.2.2.2 Attributes

No tag definitions are defined for this stereotype

#### A.2.2.3 Constraints

According to the semantics of ODP, an *object template* should contain all the information required to instantiate it

**context** IV_ObjectTemplate **inv:** self.baseClass.isAbstract = false

#### A.2.2.4 Semantics

An «IV_ObjectTemplate» Class is mapped to an *Information object template.* The isAbstract attribute is set to false.

#### A.2.2.5 Notation

UML standard syntax for Class with stereotype is used.

#### A.2.2.6 References

RM-ODP: *Object* [Part 2 – 8.1], <X> *Template* [Part 2 – 9.1.1], Information language [Part 3 – 6].

UML: Class (from Kernel) [UML – 7.3.7]

### A.2.3 Information object type

#### A.2.3.1 «IV_ObjectType»

The stereotype «IV_ObjectType» extends the metaclass Class with multiplicity [0..1]. It is intended to capture the semantics of an *object type* in the RM-ODP information language.

#### A.2.3.2 Attributes

No tag definitions are defined for this stereotype

#### A.2.3.3 Constraints

No constraints are defined for this stereotype

#### A.2.3.4 Semantics

A «IV_ObjectType» Class is mapped to an *Information object type*. In UML, a class describes a set of objects that share the same specifications of features, constraints, and semantics.

NOTE – The UML concept of class is different to the ODP concept of *class*. A UML class is a "description" of a set of objects, while an ODP class is the set of objects itself. Therefore, the UML concept of class is closer to the ODP concept of type, and there is no UML concept corresponding to the ODP concept of class. Therefore, no mapping for the ODP concept of class is provided.

#### A.2.3.5 Notation

UML standard syntax for Class with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| «IV_ObjectType» |  |

#### A.2.2.6 References

RM-ODP: *Object* [Part 2 – 8.1]; *Type* (of an <X>) [Part 2 – 9.7]; Information language [Part 3 – 6].

UML: Class (from Kernel) [UML – 7.3.7]

### A.2.4 Information action type

#### A.2.4.1 «IV_ActionType»

The stereotype «IV_ActionType» extends the metaclass Signal with multiplicity [0..1]. It is intended to capture the semantics of an action type in the RM-ODP information language.

#### A.2.4.2 Attributes

No tag definitions are defined for this stereotype

#### A.2.4.3 Constraints

No constraints are defined for this stereotype

#### A.2.4.4 Semantics

An «IV_ActionType» Signal is mapped to an *Information action type.*

In the information viewpoint, *actions* are mainly used for describing events that cause state changes, or for implementing *communications* between *objects*, i.e., flows of information.

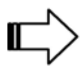
In an information specification, an *internal action* is mapped to an internal transition of a state of the state machine for the *information object* concerned.

An *interaction* is mapped to a signal sent or received by the state machines of the *information objects* concerned.

#### A.2.4.5 Notation

UML standard syntax for Signal with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| «IV_ActionType» |  |

#### A.2.4.6 References

RM-ODP: *Action* [Part 2 – 8.3]; *Type* (of an <X>) [Part 2 – 9.7]; *Information language* [Part 3 – 6]

UML: Signal (from Communications) [UML – 13.3.23]

### A.2.5 Dynamic Schema

#### A.2.5.1 «IV_Dynamic Schema»

The stereotype «IV_DynamicSchema» extends the metaclass StateMachine with multiplicity [0..1]. It is intended to capture the semantics of a *dynamic schema* in the RM-ODP information language.

#### A.2.5.2 Attributes

No tag definitions are defined for this stereotype.

#### A.2.5.3 Constraints

No constraints are defined for this stereotype.

#### A.2.5.4 Semantics

A *dynamic schema* is expressed in terms of state machines for the *information objects* in the information specification. The *actions* that relate to the state changes are mapped to signals that are sent and received on transitions of the state machines. Then, an «IV_DynamicSchema» StateMachine is mapped to a *dynamic schema.*

#### A.2.5.5 Notation

UML standard syntax for StateMachine with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| «IV_DynamicSchema» |  |

#### A.2.5.6 References

RM-ODP: *Dynamic Schema* [Part 3 – 6.1.3]; Information language [Part 3 – 6]

UML: StateMachine (from BehaviorStateMachines) [UML – 15.3.12]

### A.2.6 Static Schema

#### A.2.6.1 «IV_StaticSchema»

The stereotype «IV_StaticSchema» extends the metaclass Package with multiplicity [0..1]. It is intended to capture the semantics of a *static schema* in the RM-ODP information language.

#### A.2.6.2 Attributes

**locationInTime : date** This attribute specifies the location in time of the static schemata

#### A.2.6.3 Constraints

No constraints are defined for this stereotype

#### A.2.6.4 Semantics

An ODP *static schema* is represented as a UML package of UML objects, their attribute links, their UML link ends which have an associated target link end which is navigable, and their UML classifiers.

NOTE – The possible associations of the *information objects* described in a *static schema* with other *objects* not contemplated in the schema need not be included in the UML package, since they are not part of the specification provided by the *schema*. Therefore, whenever the absence of an association instance (i.e., a link) needs to be expressed, it should be explicitly stated (by, e.g., using constraints attached to the appropriate objects).

#### A.2.6.5 Notation

UML standard syntax for StateMachine with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| « IV_StaticSchema» |  |

#### A.2.6.6 References

RM-ODP: *Static Schema* [Part 3 – 6.1.2]; Information language [Part 3 – 6]

UML: Package (from Kernel) [UML – 7.3.37]

### A.2.7 Invariant Schema

#### A.2.7.1 «IV_InvariantSchema»

The stereotype «IV_InvariantSchema» extends the metaclasses Package and Constraint with multiplicity [0..1]. It is intended to capture the semantics of an *invariant schema* in the RM-ODP information language.

#### A.2.7.2 Attributes

No tag definitions are defined for this stereotype.

#### A.2.7.3 Constraints

No constraints are defined for this stereotype.

#### A.2.7.4 Semantics

*Invariant schemata* may impose different kinds of constraints in an information specification.

First, *invariant schemata* might provide the specification of the *types* of one or more *information objects*, that will always be satisfied by whatever behaviour the *objects* might exhibit.

This kind of *invariant schema* may be represented in a UML Package, and drawn in a class diagram, which specifies a set of *information object types* (in terms of the set of UML classes that represent such *object types*), their possible relationships (represented as UML associations), and constraints on those *object types*, on their relationships, and possibly on their *behaviours* (represented by the specification of the corresponding UML objects' state machines). The association multiplicities and the UML constraints on the different modelling elements will constrain the possible states and state changes of the UML elements to which they apply.

NOTE – OCL is the recommended notation for expressing the constraints on the modelling elements that form part of the UML representation of an invariant schema. However, other notations can be used when OCL does not provide enough expressive power, or is not appropriate due to the kind of expected user of the specification. For example, a temporal logic formula or an English text can be used for expressing a constraint that imposes some kind of fairness requirement on the behaviour of the system (e.g., "Objects of class X will produce requests to objects of class Y, no later than a given time T after condition A on objects of classes X, Y and Z is satisfied").

As noted in ODP there are cases, however, in which an *invariant schema* in an information viewpoint specification is defined over a set of concrete *information objects*. Such kind of invariant schema may be represented as a UML package of UML objects. The UML constraints on these objects, together with the specifications of the UML classifiers of these objects, constrain the possible states and state changes of the UML objects.

NOTE – The UML classifiers of the objects will constrain the possible states and state changes of the UML objects to which they apply (through the UML associations, state machines, and constraints of these classifiers).

Finally, individual UML constraints can also be used to capture *invariant schemata*.

Thus, both a «IV_InvariantSchema» Package and a «IV_InvariantSchema» Constraint can be mapped to *invariant schemata.*

#### A.2.7.5 Notation

UML standard syntax for Package or Constraint with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| «IV_InvariantSchema» |  |

#### A.2.7.6 References

RM-ODP: *Invariant Schema* [Part 3 – 6.1.1]; Information language [Part 3 – 6].

UML: Package (from Kernel) [UML – 7.3.37]; Constraint (from Kernel) [UML – 7.3.10].

### A.2.8 Information specification

#### A.2.8.1 «Information_Spec»

The stereotype Information_Spec extends the metaclass Model with multiplicity [0..1]. It is intended to capture the semantics of a *static schema* in the RM-ODP information language.

#### A.2.8.2 Attributes

No tag definitions are defined for this stereotype.

#### A.2.8.3 Constraints

No constraints are defined for this stereotype.

#### A.2.8.4 Semantics

All the UML elements representing the information specification will be defined within a UML model, stereotyped «Information_Spec». Such a model contains the UML packages that represent the *invariant*, *static* and *dynamic schemata* of the *system*. The structure of the «Information_Spec» model is detailed in clause 8.3.

#### A.2.8.5 Notation

UML standard syntax for Model with stereotype is used.

NOTE – The following icon may be used.

| | |
|---|---|
| «Information_Spec» |  |

#### A.2.8.6 References

RM-ODP: Information language [Part 3 – 6]

UML: Model (from Models) [UML – 17.3.1]

## A.3 Computational viewpoint

### A.3.1 Computational object template

#### A.3.1.1 «CV_ObjectTemplate»

The stereotype «CV_ObjectTemplate» extends the metaclass Component with multiplicity [0..1]. It is intended to capture the semantics of template for computational object in the RM-ODP computational language.

#### A.3.1.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.1.3 Constraints

– The isIndirectlyInstantiated attribute is set to true.

– A Component representing a *computational object template* has Ports and Interfaces for *interaction* with other *computational objects*.

#### A.3.1.4 Semantics

A «CV_ObjectTemplate» Component is mapped to a *Computational object template.* A UML component represents a modular part of a system which encapsulates its contents, and defines its behaviour in terms of provided and required interfaces through its ports.

The attribute isIndirectlyInstantiated of the component stereotyped «IV_ObjectTemplate» constraints the he kind of instantiation that applies to a UML component. If false, the component is instantiated as an addressable object. If true (default vuale), the component is defined at design-time, but at runtime (or execution-time) an object specified by the component does not exist, that is, the component is instantiated indirectly, through the instantiation of its realizing classifiers or parts.

#### A.3.1.5 Notation

UML standard syntax for Component with stereotype is used.

#### A.3.1.6 References

RM-ODP: *Object* [Part 2 – 8.1]; <X> *Template* [Part 2 – 9.1.1]; Computational language [Part 3 – 7]

UML: Component (from BasicComponents and PackagingComponents) [UML – 8.3.1]

### A.3.2 Computational object

#### A.3.2.1 «CV_Object»

The stereotype «CV_Object» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of *computational object* in the RM-ODP computational language.

#### A.3.2.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.2.3 Constraints

The InstanceSpecification is an instance of Component:

**context** CV_Object **inv** ComponentInstance:
self.baseInstanceSpecification.classifier->includes(Component)

#### A.3.2.4 Semantics

A «CV_Object» InstanceSpecification is mapped to a *Computational object*.

#### A.3.2.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

#### A.3.2.6 References

RM-ODP: Object [Part 2 – 8.1]; Computational language [Part 3 – 7]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.3.3 Binding object

#### A.3.3.1 «CV_BindingObject»

The stereotype «CV_BindingObject» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of *binding object* in the RM-ODP computational language. Since a *binding object* is a particular kind of *computational object*, stereotype «CV_BindingObject» inherits from «CV_Object».

#### A.3.3.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.3.3 Constraints

A *binding object* is associated with at least two different *objects*.

A *binding object* binds two or more *objects* through the same *type* of interface (*signal*, *announcement*, *interrogation*, or *flow*).

#### A.3.3.4 Semantics

A «CV_BindingObject» InstanceSpecification is mapped to a *Binding object*.

#### A.3.3.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

#### A.3.3.6 References

RM-ODP: *Object* [Part 2 – 8.1]; *Binding object* [Part 3 – 7.1.14]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.3.4 Environment contract

#### A.3.4.1 «CV_EnvironmentContract»

The stereotype «CV_EnvironmentContract» extends metaclasses Constraint and Package with multiplicity [0..1]. It is intended to capture the semantics of *environment contract* in the RM-ODP computational language.

#### A.3.4.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.4.3 Constraints

No constraints are defined for this stereotype

#### A.3.4.4 Semantics

An *environment contract* is modelled as a UML package stereotyped «CV_EnvironmentContract» when representing a set of structural and behavioural constraints between a *computational object* and its *environment*, including quality of service or other kinds of requirements. In addition, individual constraints applied to UML model elements can also be stereotyped as «CV_EnvironmentContract» when they capture such kinds of restrictions. Thus, a «CV_EnvironmentContract» Package or a «CV_EnvironmentContract» Constraint is mapped to an *Environment contract*.

#### A.3.4.5 Notation

UML standard syntax for Package or Constraint with stereotype is used.

#### A.3.4.6 References

RM-ODP: *Environment contract* [Part 2 – A.5.3]

UML: Package (from Kernel) [UML – 7.3.37]; Constraint (from Kernel) [UML – 7.3.10]

### A.3.5 Signal

#### A.3.5.1 «CV_Signal»

The stereotype «CV_Signal» extends the metaclass Message with multiplicity [0..1]. It is intended to capture the semantics of *signal* in the RM-ODP computational language.

#### A.3.5.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.5.3 Constraints

No constraints are defined for this stereotype

#### A.3.5.4 Semantics

A «CV_Signal» Message is mapped to a *Signal*, which is sent by *initiating object* and received by a *responding object*. Any *signal* should be associated with a *Signal Interface*

#### A.3.5.5 Notation

UML standard syntax for Message with stereotype is used.

#### A.3.5.6 References

RM-ODP: *Signal* [Part 3 – 7.1.1]

UML: Message (from BasicInteractions) [UML – 14.3.20]

### A.3.6 Announcement

#### A.3.6.1 «CV_Announcement»

The stereotype «CV_Announcement»extends the metaclass Message with multiplicity [0..1]. It is intended to capture the semantics of *announcement* in the RM-ODP computational language.

#### A.3.6.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.6.3 Constraints

No constraints are defined for this stereotype

#### A.3.6.4 Semantics

A «CV_Annoucement» Message is mapped to an *Announcement*, which is sent by a *client object* and received by a *server object* with no response expected. Any *announcement* should be associated with an *operation interface*.

#### A.3.6.5 Notation

UML standard syntax for Message with stereotype is used.

#### A.3.6.6 References

RM-ODP: *Announcement* [Part 3 – 7.1.3]

UML: Message (from BasicInteractions) [UML – 14.3.20]

### A.3.7 Invocation

#### A.3.7.1 «CV_Invocation»

The stereotype «CV_Invocation» extends the metaclass Message with multiplicity [0..1]. It is intended to capture the semantics of *Invocation* in the RM-ODP computational language.

#### A.3.7.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.7.3 Constraints

No constraints are defined for this stereotype

#### A.3.7.4 Semantics

A «CV_Invocation» Message is mapped to an *Invocation*. An invocation should be associated with an Operation interface.

#### A.3.7.5 Notation

UML standard syntax for Message with stereotype is used.

#### A.3.7.6 References

RM-ODP: *Announcement* [Part 3 – 7.1.3]; *Interrogation* [Part 3 – 7.1.4]

UML: Message (from BasicInteractions) [UML – 14.3.20]

### A.3.8 Termination

#### A.3.8.1 «CV_Termination»

The stereotype «CV_Termination» extends the metaclass Message with multiplicity [0..1]. It is intended to capture the semantics of *Termination* in the RM-ODP computational language.

#### A.3.8.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.8.3 Constraints

No constraints are defined for this stereotype

#### A.3.8.4 Semantics

A «CV_Termination» Message is mapped to a *Termination*. A *termination* should be associated with an *Operation Interface*.

#### A.3.8.5 Notation

UML standard syntax for Message with stereotype is used.

#### A.3.8.6 References

RM-ODP: *Interrogation* [Part 3 – 7.1.4]

UML: Message (from BasicInteractions) [UML – 14.3.20]

### A.3.9 Flow

#### A.3.9.1 «CV_Flow»

The stereotype «CV_Flow» extends metaclasses Message and Interaction with multiplicity [0..1]. It is intended to capture the semantics of *Flow* in the RM-ODP computational language.

#### A.3.9.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.9.3 Constraints

No constraints are defined for this stereotype

#### A.3.9.4 Semantics

A «CV_Flow» Message is mapped to a *Flow*. The UML message representing the *flow* is sent by *producer object* and received by *consumer object*. Also, a «CV_Flow» Interaction is mapped to a *Flow*. The *interaction* is between between the *producer* and the *consumer objects*.

#### A.3.9.5 Notation

UML standard syntax for Message or Interaction with stereotype is used.

#### A.3.9.6 References

RM-ODP: *Flow* [Part 3 – 7.1.5]

UML: Message (from BasicInteractions) [UML – 14.3.20]; Interaction (from BasicInteraction, Fragments) [UML – 14.3.13]

### A.3.10 Computational interface

#### A.3.10.1 «CV_Interface» and «CV_InterfaceTemplate»

Stereotype «CV_InterfaceTemplate» extends the metaclass Port with multiplicity [0..1]. It is intended to capture the semantics of *Computational interface template* in the RM-ODP computational language.

Stereotype «CV_Interface» extends the metaclass Port with multiplicity [0..1]. It is intended to capture the semantics of a *computational interface* in the RM-ODP computational language.

#### A.3.10.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.10.3 Constraints

*Computational interface* will be mapped to«CV_Interface» ports of UML components instances only.

*Computational interface templates* will be mapped to «CV_InterfaceTemplate» ports of UML components.

#### A.3.10.4 Semantics

A «CV_InterfaceTemplate» Port is mapped to a *Computational interface template*. The isService attribute is set to true, and the isBehaviour attribute is set to false.

A *computational interface template* is a specification, and therefore «CV_InterfaceTemplate» ports are used with UML components, not with component instances. «CV_Interface» ports of those component instances will represent the *computational interfaces* instantiated from the corresponding *interface templates*.

#### A.3.10.5 Notation

UML standard syntax for Port with stereotype is used.

#### A.3.10.6 References

RM-ODP: Interface [Part 2 – 8.4]; Computational language [Part 3 – 7]

UML: Port (from Ports) [UML – 9.3.11]

### A.3.11 Signal interface

#### A.3.11.1 «CV_SignalInterfaceTemplate» and «CV_SignalInterface»

The stereotype «CV_SignalInterfaceTemplate» extends the metaclass Port with multiplicity [0..1]. It is intended to capture the semantics of *Signal interface template* in the RM-ODP computational language.

The stereotype «CV_SignalInterface» extends the metaclass Port with multiplicity [0..1]. It is intended to capture the semantics of *Signal interface* in the RM-ODP computational language.

#### A.3.11.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.11.3 Constraints

*Signal interfaces* will be mapped to«CV_SignalInterface» ports of UML components instances.

*Signal interface* templates will be mapped to «CV_SignalInterfaceTemplate» ports of UML components.

#### A.3.11.4 Semantics

A «CV_SignalInterfaceTemplate» Port is mapped to a *Signal interface template*. The isService attribute is set to true, and the isBehaviour attribute is set to false.

A *signal interface template* is a specification, and therefore «CV_SignalInterfaceTemplate» ports are used with UML components, not with component instances. «CV_SignalInterface» ports of those component instances will represent the *signal interfaces* instantiated from the corresponding *interface templates*.

#### A.3.11.5 Notation

UML standard syntax for Port with stereotype is used.

#### A.3.11.6 References

RM-ODP: *Signal interface* [Part 3– 7.1.6]

UML: Port (from Ports) [UML – 9.3.11]

### A.3.12 Operation interface

#### A.3.12.1 «CV_OperationInterface» and «CV_OperationTemplate»

The stereotype «CV_OperationInterfaceTemplate» extends the metaclass Port with multiplicity [0..1]. It is intended to capture the semantics of *Operation interface template* in the RM-ODP computational language.

The stereotype «CV_OperationInterface» extends the metaclass Port with multiplicity [0..1]. It is intended to capture the semantics of *Operation interface* in the RM-ODP computational language.

#### A.3.12.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.12.3 Constraints

*Operation interfaces* will be mapped to«CV_OperationInterface» ports of UML components instances.

*Operation interface* templates will be mapped to «CV_OperationInterfaceTemplate» ports of UML components.

#### A.3.12.4 Semantics

A «CV_OperationInterface» Port is mapped to an *Operation interface*. The isService attribute is set to true, and the isBehaviour attribute is set to false.

An *operation interface template* is a specification, and therefore «CV_OperationInterfaceTemplate» ports are used with UML components, not with component instances. «CV_OperationInterface» ports of those component instances will represent the *operation interfaces* instantiated from the corresponding *interface templates*.

#### A.3.12.5 Notation

UML standard syntax for Port with stereotype is used.

#### A.3.12.6 References

RM-ODP: *Operation interface* [Part 3– 7.1.7]

UML: Port (from Ports) [UML – 9.3.11]

### A.3.13 Stream interface

#### A.3.13.1 «CV_StreamInterface» and «CV_StreamInterfaceTemplate»

The stereotype «CV_StreamInterfaceTemplate» extends the metaclass Port with multiplicity [0..1]. It is intended to capture the semantics of *Stream interface template* in the RM-ODP computational language.

The stereotype «CV_StreamInterface» extends the metaclass Port with multiplicity [0..1]. It is intended to capture the semantics of *Stream interface* in the RM-ODP computational language.

#### A.3.13.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.13.3 Constraints

*Stream interfaces* will be mapped to«CV_ StreamInterface» ports of UML components instances.

*Stream interface* templates will be mapped to «CV_ StreamInterfaceTemplate» ports of UML components.

#### A.3.13.4 Semantics

A «CV_StreamInterface» Port is mapped to an *Stream interface*. The isService attribute is set to true, and the isBehaviour attribute is set to false.

A *stream interface template* is a specification, and therefore «CV_StreamInterfaceTemplate» ports are used with UML components, not with component instances. «CV_StreamInterface» ports of those component instances will represent the *stream interfaces* instantiated from the corresponding *interface templates*.

#### A.3.13.5 Notation

UML standard syntax for Port with stereotype is used.

#### A.3.13.6 References

RM-ODP: *Stream interface* [Part 3– 7.1.8]

UML: Port (from Ports) [UML – 9.3.11]

### A.3.14 Computational interface signature

#### A.3.14.1 «CV_InterfaceSignature»

The stereotype «CV_Signature» extends the metaclass Interface with multiplicity [0..1]. It is intended to capture the semantics of *Computational interface signature* in the RM-ODP computational language.

#### A.3.14.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.14.3 Constraints

No constraints are defined for this stereotype

#### A.3.14.4 Semantics

A «CV_InterfaceSignature» Interface is mapped to a *Computational interface signature*.

#### A.3.14.5 Notation

UML standard syntax for Interface with stereotype is used.

#### A.3.14.6 References

RM-ODP: *Interface signature* [Part 2 – 9.12]

UML: Interface (from Interfaces) [UML – 7.3.24]

### A.3.15 Signal interface signature

#### A.3.15.1 «CV_SignalInterfaceSignature»

The stereotype «CV_SignalInterfaceSignature» extends the metaclass Interface with multiplicity [0..1]. It is intended to capture the semantics of *Signal interface signature* in the RM-ODP computational language.

#### A.3.15.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.15.3 Constraints

Associated Interfaces are Signal Interfaces (see A.3.11.3)

#### A.3.15.4 Semantics

A «CV_SignalInterfaceSignature» Interface is mapped to a *Signal interface signature*.

#### A.3.15.5 Notation

UML standard syntax for Interface with stereotype is used.

#### A.3.15.6 References

RM-ODP: *Signal interface signature* [Part 3 – 7.1.11]

UML: Interface (from Interfaces) [UML – 7.3.24]

### A.3.16 Operation interface signature

#### A.3.16.1 «CV_OperationInterfaceSignature»

The stereotype «CV_OperationInterfaceSignature» extends the metaclass Interface with multiplicity [0..1]. It is intended to capture the semantics of *Operation interface signature* in the RM-ODP computational language.

#### A.3.16.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.16.3 Constraints

Associated Interfaces are Operation Interfaces (see A.3.12.3)

#### A.3.16.4 Semantics

A «CV_OperationInterfaceSignature» Interface is mapped to an *Operation interface signature*.

#### A.3.16.5 Notation

UML standard syntax for Interface with stereotype is used.

#### A.3.16.6 References

RM-ODP: *Operation interface signature* [Part 3 – 7.1.12]

UML: Interface (from Interfaces) [UML – 7.3.24]

### A.3.17 Stream interface signature

#### A.3.17.1 «CV_StreamInterfaceSignature»

The stereotype «CV_StreamInterfaceSignature» extends the metaclass Interface with multiplicity [0..1]. It is intended to capture the semantics of *Stream interface signature* in the RM-ODP computational language.

#### A.3.17.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.17.3 Constraints

No constraints are defined for this stereotype

#### A.3.17.4 Semantics

A «CV_StreamInterfaceSignature» Interface is mapped to a *Stream interface signature*.

#### A.3.17.5 Notation

UML standard syntax for Interface with stereotype is used.

#### A.3.17.6 References

RM-ODP: *Stream interface signature* [Part 3 – 7.1.13]

UML: Interface (from Interfaces) [UML – 7.3.24]

### A.3.18 Computational signature

*Computational signatures* can be modelled as UML receptions, UML operations, or interfaces. UML receptions will be used to specify *signatures* of *computational interactions* which are expressed as individual *signals* (*signals*, *announcements*, *invocations* and *terminations*). UML operations can be used to map ODP *interrogation signatures* that are composed of an *invocation signature* and a *termination signature*. Finally, UML interfaces will be used for mapping *flow signatures*, when *flows* are expressed in terms of sequences of *signals*.

### A.3.19 Signal signature

#### A.3.19.1 «CV_SignalSignature»

The stereotype «CV_SignalSignature» extends the metaclass Reception with multiplicity [0..1]. It is intended to capture the semantics of *Signal signature* in the RM-ODP computational language.

#### A.3.19.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.19.3 Constraints

No constraints are defined for this stereotype

#### A.3.19.4 Semantics

A «CV_SignalSignature» Reception is mapped to a *Signal signature*.

#### A.3.19.5 Notation

UML standard syntax for Reception with stereotype is used.

#### A.3.19.6 References

RM-ODP: *Signal signature* [Part 3 – 7.1.11]

UML: Reception (from Communications) [UML – 13.3.22]

### A.3.20 Announcement signature

#### A.3.20.1 «CV_AnnouncementSignature»

The stereotype «CV_AnnouncementSignature» extends the metaclass Reception ith multiplicity [0..1]. It is intended to capture the semantics of *Announcement signature* in the RM-ODP computational language.

#### A.3.20.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.20.3 Constraints

No constraints are defined for this stereotype

#### A.3.20.4 Semantics

A «CV_AnnouncementSignature» Reception is mapped to an *Announcement signature*.

#### A.3.20.5 Notation

UML standard syntax for Reception with stereotype is used.

#### A.3.20.6 References

RM-ODP: *Operation interface signature* [Part 3 – 7.1.12]

UML: Reception (from Communications) [UML – 13.3.22]

### A.3.21 Invocation signature

#### A.3.21.1 «CV_InvocationSignature»

The stereotype «CV_InvocationSignature» extends the metaclass Reception with multiplicity [0..1]. It is intended to capture the semantics of *Invocation signature* in the RM-ODP computational language.

#### A.3.21.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.21.3 Constraints

No constraints are defined for this stereotype

#### A.3.21.4 Semantics

A «CV_InvocationSignature» Reception is mapped to an *Interrogation signature*.

#### A.3.21.5 Notation

UML standard syntax for Reception with stereotype is used.

#### A.3.21.6 References

RM-ODP: *Operation interface signature* [Part 3 – 7.1.12]

UML: Reception (from Communications) [UML – 13.3.22]

### A.3.22 Termination signature

#### A.3.22.1 «CV_TerminationSignature»

The stereotype «CV_TerminationSignature» extends the metaclass Reception with multiplicity [0..1]. It is intended to capture the semantics of *Termination signature* in the RM-ODP computational language.

#### A.3.22.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.22.3 Constraints

No constraints are defined for this stereotype

#### A.3.22.4 Semantics

A «CV_TerminationSignature» Reception is mapped to a *Termination signature*.

#### A.3.22.5 Notation

UML standard syntax for Reception with stereotype is used.

#### A.3.22.6 References

RM-ODP: *Operation interface signature* [Part 3 – 7.1.12]

UML: Reception (from Communications) [UML – 13.3.22]

### A.3.23 Interrogation signature

#### A.3.23.1 «CV_InterrogationSignature»

The stereotype «CV_InterrogationSignature» extends the metaclass Operation with multiplicity [0..1]. It is intended to capture the semantics of an *interrogation signature* in the RM-ODP computational language.

#### A.3.23.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.23.3 Constraints

No constraints are defined for this stereotype

#### A.3.23.4 Semantics

An *interrogation signature* is a *signature* for an *interrogation*, when it is composed of an *invocation* and a *termination*. A «CV_ InterrogationSignature» Operation is mapped to an *Interrogation signature*. This stereotyped UML operation represents an *action template* which includes name for the *invocation*, the number, names and *types* of its *parameters*, the indication of *client* or *server*, and the number, names and *types* of the *termination's parameters*.

#### A.3.23.5 Notation

UML standard syntax for Operation with stereotype is used.

#### A.3.23.6 References

RM-ODP: *Operation interface signature* [Part 3 – 7.1.12]

UML: Operation (from Communications) [UML – 13.3.21]

### A.3.24 Flow signature

#### A.3.24.1 «CV_FlowSignature»

The stereotype «CV_FlowSignature» extends the metaclass Interface with multiplicity [0..1]. It is intended to capture the semantics of *Flow signature* in the RM-ODP computational language.

#### A.3.24.2 Attributes

No tag definitions are defined for this stereotype

#### A.3.24.3 Constraints

No constraints are defined for this stereotype

#### A.3.24.4 Semantics

A «CV_FlowSignature» Interface is mapped to a *Flow signature*. This stereotyped UML interface represents an *action template* which includes name for the *flow*, the number, names and *types* of its associated *signals* and their *parameters*, and an indication of *producer* or *consumer*.

#### A.3.24.5 Notation

UML standard syntax for Interface with stereotype is used.

#### A.3.24.6 References

RM-ODP: *Stream interface signature* [Part 3 – 7.1.13]

UML: Interface (from Interfaces) [UML – 7.3.24]

## A.4 Engineering viewpoint

### A.4.1 Engineering object template

#### A.4.1.1 «NV_ObjectTemplate»

The stereotype «NV_ObjectTemplate» extends the metaclass Component with multiplicity [0..1]. It is intended to capture the semantics of *template* for *engineering object* in the RM-ODP engineering language.

#### A.4.1.2 Attributes

| | |
|---|---|
| **deployedNode: String** | Defines a reference to a *node* where an *engineering object* is deployed. |
| **securityDomain: String** | Defines a reference of a *security domain* it may belong. |
| **managementDomain: String** | Defines a reference of a *management domain* it may belong. |

#### A.4.1.3 Constraints

The isIndirectlyInstantiated attribute is set to false.

**context** NV_ObjectTemplate **inv**:
self.baseComponent.isIndirectlyInstantiated = false

#### A.4.1.4 Semantics

A «NV_ObjectTemplate» Component is mapped to a *Component*. The isIndirectlyInstantiated attribute is set to false. This attribute constraints the he kind of instantiation that applies to a UML component. If false, the component is instantiated as an addressable object.

#### A.4.1.5 Notation

UML standard syntax for Component with stereotype is used.

#### A.4.1.6 References

RM-ODP: *Object* [Part 2 – 8.1], <X> *Template* [Part 2 – 9.1.1], Engineering language [Part 3 – 8]

UML: Component (from BasicComponents and PackagingComponents) [UML – 8.3.1]

### A.4.2 Engineering object

#### A.4.2.1 «NV_Object»

The stereotype «NV_Object» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of *engineering object* in the RM-ODP engineering language.

#### A.4.2.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.2.3 Constraints

The InstanceSpecification is an instance of Component.

self.baseInstanceSpecification.classifier->includes(Component)

#### A.4.2.4 Semantics

An «NV_Object» InstanceSpecification is mapped to an *engineering object*.

#### A.4.2.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

#### A.4.2.6 References

RM-ODP: *Object* [Part 2 – 8.1], Engineering language [Part 3 – 8]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.4.3 Basic engineering object

#### A.4.3.1 «NV_BEO»

The stereotype «NV_BEO» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of *basic engineering object* in the RM-ODP engineering language.

#### A.4.3.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.3.3 Constraints

The InstanceSpecification is an instance of Component.

self.baseInstanceSpecification.classifier->includes(Component)

#### A.4.3.4 Semantics

An «NV_BEO» InstanceSpecification is mapped to a *Basic engineering object*.

#### A.4.3.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

#### A.4.3.6 References

RM-ODP: *Basic engineering object* [Part 3 – 8.1.1]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.4.4 Cluster

#### A.4.4.1 «NV_Cluster»

The stereotype «NV_Cluster» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of *Cluster* in the RM-ODP engineering language.

#### A.4.4.2 Attributes

No tag definitions are defined for this stereotype.

#### A.4.4.3 Constraints

The InstanceSpecification is an instance of Component.

    self.baseInstanceSpecification.classifier->includes(Component)

#### A.4.4.4 Semantics

An «NV_Cluster» InstanceSpecification is mapped to a *Cluster*.

#### A.4.4.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

#### A.4.4.6 References

RM-ODP: *Cluster* [Part 3 – 8.1.3]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.4.5 Cluster manager

#### A.4.5.1 «NV_ClusterManager»

The stereotype «NV_ClusterManager» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of *Cluster manager* in the RM-ODP engineering language.

#### A.4.5.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.5.3 Constraints

The InstanceSpecification is an instance of Component.

    self.baseInstanceSpecification.classifier->includes(Component)

#### A.4.5.4 Semantics

An «NV_ClusterManager» InstanceSpecification is mapped to a *Cluster manager*.

#### A.4.5.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

#### A.4.5.6 References

RM-ODP: *Cluster manager* [Part 3 – 8.1.3]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.4.6 Capsule

#### A.4.6.1 «NV_Capsule»

The stereotype «NV_Capsule» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of *Capsule* in the RM-ODP engineering language.

#### A.4.6.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.6.3 Constraints

The InstanceSpecification is an instance of Component.

self.baseInstanceSpecification.classifier->includes(Component)

#### A.4.6.4 Semantics

An «NV_Capsule» InstanceSpecification is mapped to a *Capsule*.

#### A.4.6.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

#### A.4.6.6 References

RM-ODP: *Capsule* [Part 3 – 8.1.4]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.4.7 Capsule manager

#### A.4.7.1 «NV_CapsuleManager»

The stereotype «NV_CapsuleManager» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of *Capsule manager* in the RM-ODP engineering language.

#### A.4.7.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.7.3 Constraints

The InstanceSpecification is an instance of Component.

self.baseInstanceSpecification.classifier->includes(Component)

#### A.4.7.4 Semantics

An «NV_CapsuleManager» InstanceSpecification is mapped to a *Capsule manager*.

#### A.4.7.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

#### A.4.7.6 References

RM-ODP: *Capsule manager* [Part 3 – 8.1.5]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.4.8 Nucleus

#### A.4.8.1 «NV_Nucleus»

The stereotype «NV_Nucleus» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of *Nucleus* in the RM-ODP engineering language.

#### A.4.8.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.8.3 Constraints

The InstanceSpecification is an instance of Component.

self.baseInstanceSpecification.classifier->includes(Component)

#### A.4.8.4 Semantics

An «NV_Nucleus» InstanceSpecification is mapped to a *Nucleus*.

#### A.4.8.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

#### A.4.8.6 References

RM-ODP: *Nucleus* [Part 3 – 8.1.6]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.4.9 Node

#### A.4.9.1 «NV_Node»

The stereotype «NV_Node» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of *Node* in the RM-ODP engineering language.

#### A.4.9.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.9.3 Constraints

The InstanceSpecification is an instance of Component.

```
self.baseInstanceSpecification.classifier->includes(Component)
```

#### A.4.9.4 Semantics

An «NV_Node» InstanceSpecification is mapped to a *Node.*

#### A.4.9.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

#### A.4.9.6 References

RM-ODP: *Node* [Part 3 – 8.1.7]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.4.10 Channel

#### A.4.10.1 «NV_Channel»

The stereotype «NV_Channel» extends the metaclass Package with multiplicity [0..1]. It is intended to capture the semantics of *Channel* in the RM-ODP engineering language.

#### A.4.10.2 Attributes

#### A.4.10.3 Constraints

The only elements that a *channel* may contain are *binders*, *stubs*, *protocol objects* and *interceptors*.

#### A.4.10.4 Semantics

An «NV_Channel» Package is mapped to a *Channel.*

#### A.4.10.5 Notation

UML standard syntax for Package with stereotype is used.

#### A.4.10.6 References

RM-ODP: 3-8.1.8 *Channel* [Part 3 – 8.1.8]

UML: Package (from Kernel) [UML – 7.3.37]

### A.4.11 Stub

#### A.4.11.1 «NV_Stub»

The stereotype «NV_Stub» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of *Stub* in the RM-ODP engineering language.

#### A.4.11.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.11.3 Constraints

The InstanceSpecification is an instance of Component.

self.baseInstanceSpecification.classifier->includes(Component)

#### A.4.11.4 Semantics

An «NV_Stub» InstanceSpecification is mapped to a *Stub*.

#### A.4.11.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

#### A.4.11.6 References

RM-ODP: *Stub* [Part 3 – 8.1.9]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.4.12 Binder

#### A.4.12.1 «NV_Binder»

The stereotype «NV_Binder» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of *Binder* in the RM-ODP engineering language.

#### A.4.12.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.12.3 Constraints

The InstanceSpecification is an instance of Component.

self.baseInstanceSpecification.classifier->includes(Component)

#### A.4.12.4 Semantics

An «NV_Binder» InstanceSpecification is mapped to a *Binder*.

#### A.4.12.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

#### A.4.12.6 References

RM-ODP: *Binder* [Part 3 – 8.1.10]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.4.13 <X> Interceptor

#### A.4.13.1 «NV_Interceptor»

The stereotype «NV_Interceptor» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of *Interceptor* in the RM-ODP engineering language.

#### A.4.13.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.13.3 Constraints

The InstanceSpecification is an instance of Component.

self.baseInstanceSpecification.classifier->includes(Component)

#### A.4.13.4 Semantics

An «NV_Interceptor» InstanceSpecification is mapped to an *Interceptor*.

#### A.4.13.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

#### A.4.13.6 References

RM-ODP: <X> *Interceptor* [Part 3 – 8.1.11]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.4.14 Protocol object

#### A.4.14.1 «NV_ProtocolObject»

The stereotype «NV_ProtocolObject» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of *Protocol object* in the RM-ODP engineering language.

#### A.4.14.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.14.3 Constraints

The InstanceSpecification is an instance of Component.

self.baseInstanceSpecification.classifier->includes(Component)

#### A.4.14.4 Semantics

An «NV_ProtocolObject» InstanceSpecification is mapped to a *Protocol object.*

#### A.4.14.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

#### A.4.14.6 References

RM-ODP: *Protocol object* [Part 3 – 8.1.12]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.4.15 CommunicationDomain

#### A.4.15.1 «NV_CommunicationDomain»

The stereotype «NV_CommunicationDomain» extends the metaclass Package with multiplicity [0..1]. It is intended to capture the semantics of *Communication domain* in the RM-ODP engineering language.

#### A.4.15.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.15.3 Constraints

No constraints are defined for this stereotype

#### A.4.15.4 Semantics

An «NV_CommunicationDomain» Package is mapped to a *Communication domain.*

#### A.4.15.5 Notation

UML standard syntax for Package with stereotype is used.

#### A.4.15.6 References

RM-ODP: *Communication domain* [Part 3 – 8.1.13]

UML: Package (from Kernel) [UML – 7.3.37]

### A.4.16 Communication interface

#### A.4.16.1 «NV_ CommunicationInterface»

The stereotype «NV_CommunicationInterface» extends the metaclass Port with multiplicity [0..1]. It is intended to capture the semantics of *Communication interface* in the RM-ODP engineering language.

#### A.4.16.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.16.3 Constraints

No constraints are defined for this stereotype

#### A.4.16.4 Semantics

An «NV_CommunicationInterface» Port is mapped to a *Communication interface*. The isService attribute is set to true, and the isBehaviour attribute is set to false.

#### A.4.16.5 Notation

UML standard syntax for Port with stereotype is used.

#### A.4.16.6 References

RM-ODP: *Communication interface* [Part 3 – 8.1.14]

UML: Port (from Ports) [UML – 9.3.11]

### A.4.17 Binding endpoint identifier

#### A.4.17.1 «NV_ BindingEndpointIdentifier»

The stereotype «NV_BindingEndpointIdentifier» extends the metaclass ValueSpecification with multiplicity [0..1]. It is intended to capture the semantics of *Binding endpoint identifier* in the RM-ODP engineering language.

#### A.4.17.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.17.3 Constraints

No constraints are defined for this stereotype

#### A.4.17.4 Semantics

An «NV_BindingEndpointIdentifier» ValueSpecification is mapped to a *Binding endpoint identifier*.

#### A.4.17.5 Notation

UML standard syntax for ValueSpecification with stereotype is used.

#### A.4.17.6 References

RM-ODP: *Binding endpoint identifier* [Part 3 – 8.1.15]

UML: ValueSpecification (from Kernel) [UML – 7.3.54]

### A.4.18 Engineering interface reference

#### A.4.18.1 «NV_ InterfaceReference»

The stereotype «NV_InterfaceReference» extends the metaclass ValueSpecification with multiplicity [0..1]. It is intended to capture the semantics of *Engineering interface reference* in the RM-ODP engineering language.

#### A.4.18.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.18.3 Constraints

No constraints are defined for this stereotype

#### A.4.18.4 Semantics

An «NV_InterfaceReference» ValueSpecification is mapped to an *Engineering interface reference.*

#### A.4.18.5 Notation

UML standard syntax for ValueSpecification with stereotype is used.

#### A.4.18.6 References

RM-ODP: *Engineering interface reference*, *Interface References Standard* [Part 3 – 8.1.16]

UML: ValueSpecification (from Kernel) [UML – 7.3.54]

### A.4.19 Engineering interface reference management domain

#### A.4.19.1 «NV_ InterfaceReferenceManagementDomain»

The stereotype «NV_InterfaceReferenceManagementDomain» extends the metaclass Package with multiplicity [0..1]. It is intended to capture the semantics of *Engineering interface reference management domain* in the RM-ODP engineering language.

#### A.4.19.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.19.3 Constraints

No constraints are defined for this stereotype

#### A.4.19.4 Semantics

An «NV_InterfaceReferenceManagementDomain» Package is mapped to an *Engineering interface reference management domain.*

#### A.4.19.5 Notation

UML standard syntax for Package with stereotype is used.

#### A.4.19.6 References

RM-ODP: *Engineering interface reference management domain*, *Interface Reference Standard* [Part 3 – 8.1.17]

UML: Package (from Kernel) [UML – 7.3.37]

### A.4.20 Engineering interface reference management policy

#### A.4.20.1 «NV_ InterfaceReferenceManagementPolicy»

The stereotype «NV_InterfaceReferenceManagementPolicy» extends the metaclass Constraint with multiplicity [0..1]. It is intended to capture the semantics of *Engineering interface reference management policy* in the RM-ODP engineering language.

#### A.4.20.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.20.3 Constraints

No constraints are defined for this stereotype

#### A.4.20.4 Semantics

An «NV_ InterfaceReferenceManagementPolicy» Constraint is mapped to an *Engineering interface reference management policy*.

#### A.4.20.5 Notation

UML standard syntax for Constraint with stereotype is used.

#### A.4.20.6 References

RM-ODP: *Engineering interface reference management policy*, *Interface Reference Standard* [Part 3 – 8.1.18]

UML: Constraint (from Kernel) [UML – 7.3.10]

### A.4.21 Cluster template

#### A.4.21.1 «NV_ClusterTemplate»

The stereotype «NV_ClusterTemplate» extends the metaclass Component with multiplicity [0..1]. It is intended to capture the semantics of *Cluster template* in the RM-ODP engineering language.

#### A.4.21.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.21.3 Constraints

No constraints are defined for this stereotype

#### A.4.21.4 Semantics

An «NV_ClusterTemplate» Component is mapped to a *Cluster template.*

#### A.4.21.5 Notation

UML standard syntax for Component with stereotype is used.

#### A.4.21.6 References

RM-ODP: *Cluster template* [Part 3 – 8.1.19]

UML: Component (from BasicComponents and PackagingComponents) [UML – 8.3.1]

### A.4.22 Checkpoint

#### A.4.22.1 «NV_Checkpoint»

The stereotype «NV_Checkpoint» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of *Checkpoint* in the RM-ODP engineering language.

#### A.4.22.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.22.3 Constraints

The InstanceSpecification is an instance of Component.

```
self.baseInstanceSpecification.classifier->includes(Component)
```

#### A.4.22.4 Semantics

An «NV_Checkpoint» InstanceSpecification is mapped to a *Checkpoint.*

#### A.4.22.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

#### A.4.22.6 References

RM-ODP: Checkpoint [Part 3 – 8.1.20]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.4.23 Checkpointing

#### A.4.23.1 «NV_Checkpointing»

The stereotype «NV_Checkpointing» extends the metaclass Activity with multiplicity [0..1]. It is intended to capture the semantics of *Checkpointing* in the RM-ODP engineering language.

#### A.4.23.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.23.3 Constraints

No constraints are defined for this stereotype

#### A.4.23.4 Semantics

An «NV_Checkpointing» Activity is mapped to a *Checkpointing*. The isReadOnly attribute is set to false, and the isSingleExecution attribute is set to false.

#### A.4.23.5 Notation

UML standard syntax for Activity with stereotype is used.

#### A.4.23.6 References

RM-ODP: *Checkpointing* [Part 3 – 8.1.21]

UML: Activity (from BasicActivities, CompleteActivities, FundamentalActivities, StructuredActivities) [UML – 12.3.4]

### A.4.24 Cluster checkpoint

#### A.4.24.1 «NV_ ClusterCheckpoint»

The stereotype «NV_ClusterCheckpoint» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of *Cluster checkpoint* in the RM-ODP engineering language.

#### A.4.24.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.24.3 Constraints

The InstanceSpecification is an instance of Component.

    self.baseInstanceSpecification.classifier->includes(Component)

#### A.4.24.4 Semantics

An «NV_ClusterCheckpoint» InstanceSpecification is mapped to a *Cluster checkpoint.*

#### A.4.24.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

#### A.4.24.6 References

RM-ODP: *Cluster checkpoint* [Part 3 – 8.1.22]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.4.25 Deactivation

#### A.4.25.1 «NV_ Deactivation»

The stereotype «NV_Deactivation» extends the metaclasses Activity, Interface and Action with multiplicity [0..1]. It is intended to capture the semantics of *Deactivation* in the RM-ODP engineering language.

#### A.4.25.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.25.3 Constraints

No constraints are defined for this stereotype

#### A.4.25.4 Semantics

When a «NV_Deactivation» Activity is mapped to a *Deactivation*, the isReadOnly attribute is set to false, and the isSingleExecution attribute is set to false.

#### A.4.25.5 Notation

UML standard syntax for Activity with stereotype is used.

#### A.4.25.6 References

RM-ODP: *Deactivation* [Part 3 – 8.1.23]

UML: Activity (from BasicActivities, CompleteActivities, FundamentalActivities, StructuredActivities) [UML – 12.3.4]; Interface (from Interfaces) [UML – 7.3.24]; Action [UML – 12.3.2].

### A.4.26 Cloning

#### A.4.26.1 «NV_Cloning»

The stereotype «NV_Cloning» extends the metaclasses Activity, Interface and Action with multiplicity [0..1]. It is intended to capture the semantics of *Cloning* in the RM-ODP engineering language.

#### A.4.26.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.26.3 Constraints

No constraints are defined for this stereotype

#### A.4.26.4 Semantics

When a «NV_Cloning» Activity is mapped to a *Cloning*, the isReadOnly attribute is set to false, and the isSingleExecution attribute is set to false.

#### A.4.26.5 Notation

UML standard syntax for Activity with stereotype is used.

#### A.4.26.6 References

RM-ODP: *Cloning* [Part 3 – 8.1.24]

UML: Activity (from BasicActivities, CompleteActivities, FundamentalActivities, StructuredActivities) [UML – 12.3.4]; Interface (from Interfaces) [UML – 7.3.24]; Action [UML – 12.3.2].

### A.4.27 Recovery

#### A.4.27.1 «NV_Recovery»

The stereotype «NV_Recovery» extends the metaclasses Activity, Interface and Action with multiplicity [0..1]. It is intended to capture the semantics of *Recovery* in the RM-ODP engineering language.

#### A.4.27.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.27.3 Constraints

No constraints are defined for this stereotype

#### A.4.27.4 Semantics

When a «NV_Recovery» Activity is mapped to a *Recovery*, the isReadOnly attribute is set to false, and the isSingleExecution attribute is set to false.

#### A.4.27.5 Notation

UML standard syntax for Activity with stereotype is used.

#### A.4.27.6 References

RM-ODP: *Recovery* [Part 3 – 8.1.25]

UML: Activity (from BasicActivities, CompleteActivities, FundamentalActivities, StructuredActivities) [UML – 12.3.4]; Interface (from Interfaces) [UML – 7.3.24]; Action [UML – 12.3.2]

### A.4.28 Reactivation

#### A.4.28.1 «NV_Reactivation»

The stereotype «NV_Reactivation» extends the metaclasses Activity, Interface and Action with multiplicity [0..1]. It is intended to capture the semantics of *Reactivation* in the RM-ODP engineering language.

#### A.4.28.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.28.3 Constraints

No constraints are defined for this stereotype

#### A.4.28.4 Semantics

When a «NV_Reactivation» Activity is mapped to a *Reactivation*, the isReadOnly attribute is set to false, and the isSingleExecution attribute is set to false.

#### A.4.28.5 Notation

UML standard syntax for Activity with stereotype is used.

#### A.4.28.6 References

RM-ODP: *Reactivation* [Part 3 – 8.1.26]

UML: Activity (from BasicActivities, CompleteActivities, FundamentalActivities, StructuredActivities) [UML – 12.3.4]; Interface (from Interfaces) [UML – 7.3.24]; Action [UML – 12.3.2]

### A.4.29 Migration

#### A.4.29.1 «NV_Migration»

The stereotype «NV_Migration» extends the metaclasses Activity, Interface and Action with multiplicity [0..1]. It is intended to capture the semantics of *Migration* in the RM-ODP engineering language.

#### A.4.29.2 Attributes

No tag definitions are defined for this stereotype

#### A.4.29.3 Constraints

No constraints are defined for this stereotype

#### A.4.29.4 Semantics

When a «NV_Migration» Activity is mapped to a *Migration*, the isReadOnly attribute is set to false, and the isSingleExecution attribute is set to false.

#### A.4.29.5 Notation

UML standard syntax for Activity with stereotype is used.

#### A.4.29.6 References

RM-ODP: *Migration* [Part 3 – 8.1.27]

UML: Activity (from BasicActivities, CompleteActivities, FundamentalActivities, StructuredActivities) [UML – 12.3.4]; Interface (from Interfaces) [UML – 7.3.24]; Action [UML – 12.3.2]

## A.5 Technology viewpoint

### A.5.1 Technology object

#### A.5.1.1 «TV_Object»

The stereotype «TV_Object» extends the metaclass InstanceSpecification with multiplicity [0..1]. It is intended to capture the semantics of *technology object* in the RM-ODP technology language.

#### A.5.1.2 Attributes

No tag definitions are defined for this stereotype.

#### A.5.1.3 Constraints

A *technology object* is an instance of a Node or of an Artefact:

**context** IV_Object **inv:**
self.baseInstanceSpecification.classifier->includes(Node) **or**
self.baseInstanceSpecification.classifier->includes(Artefact)

#### A.5.1.4 Semantics

A «TV_Object» InstanceSpecification of a Node or an Artefact is mapped to a *Technology object*. A UML Artefact represents implementation or realization of functionality identified in its engineering viewpoint specification.

A «TV_Object» InstanceSpecification of a Node can also be mapped to a *technology object*.

#### A.5.1.5 Notation

UML standard syntax for InstanceSpecification with stereotype is used.

#### A.5.1.6 References

RM-ODP: *Object* [Part 2 – 8.1], technology language [Part 3 – 9]

UML: InstanceSpecification (from Kernel) [UML – 7.3.2]

### A.5.2 Technology object type

#### A.5.2.1 «TV_ObjectType»

The stereotype «TV_ObjectType» extends the metaclass Artefact with multiplicity [0..1]. Also, the stereotype «TV_ObjectType» extends the metaclass Node with multiplicity [0..1]. It is intended to represent the semantics of *technology object types* in RM-ODP technology language, which characterize the different kinds of *technology objects* used in the technology specifications.

#### A.5.2.2 Attributes

No tag definitions are defined for this stereotype

#### A.5.2.3 Constraints

Every *technology object type* is associated with at least one *implementable standard* (to which it conforms).

#### A.5.2.4 Semantics

An «TV_ObjectType» Artefact is mapped to a *technology object type*. A UML artifact represents implementation or realization of functionality identified in its engineering viewpoint specification.

A «TV_ObjectType» Node can also be mapped to a *technology object type*. The filename attribute is used to refer to its name. A UML Node represents a run-time computational resource, such as computer, including execution environment for deployed artifacts.

«TV_ObjectType» Artefacts or Nodes are valid classifiers for the InstanceSpecifications stereotyped «TV_Object» that model the *technology objects* of the technology specifications.

#### A.5.2.5 Notation

UML standard syntax for Artefact or Node with stereotype is used.

#### A.5.2.6 References

RM-ODP: *Object* [Part 2 – 8.1], technology language [Part 3 – 9]

UML: 10.3.1 Artifact (from Artifacts, Nodes) [UML – 10.3.1], Node (from Nodes) [UML – 10.3.11]

### A.5.3 Implementation standard

#### A.5.3.1 «TV_ ImplementationStandard»

The stereotype «TV_ImplementationStandard» extends the metaclass Componentt with multiplicity [0..1]. It is intended to capture the semantics of *Implementation standard* in the RM-ODP technology language.

#### A.5.3.2 Attributes

No tag definitions are defined for this stereotype

#### A.5.3.3 Constraints

Every *implementation standard* is associated with (or is implemented as) one or more *technology objects*.

#### A.5.3.4 Semantics

A «TV_ImplementationStandard» Component is mapped to an *Implementation standard.*

#### A.5.3.5 Notation

UML standard syntax for Component with stereotype is used.

#### A.5.3.6 References

RM-ODP: *Implementable standard* [Part 3 – 9.1.1]

UML: Component (from BasicComponents and PackagingComponents) [UML – 8.3.1]

### A.5.4 Implementation

#### A.5.4.1 «TV_ Implementation»

The stereotype «TV_Implementation» extends the metaclass Activity with multiplicity [0..1]. It is intended to capture the semantics of *Implementation* in the RM-ODP technology language.

#### A.5.4.2 Attributes

No tag definitions are defined for this stereotype

#### A.5.4.3 Constraints

Every *implementation* is associated with (or produces) one or more *technology objects*.

#### A.5.4.4 Semantics

A «TV_Implementation» Activity is mapped to an *Implementation.* The isReadOnly attribute is set to false, and the isSingleExecution attribute is set to false.

#### A.5.4.5 Notation

UML standard syntax for Activity with stereotype is used.

#### A.5.4.6 References

RM-ODP: *Implementation* [Part 3 – 9.1.2]

UML: Activity (from BasicActivities, CompleteActivities, FundamentalActivities, StructuredActivities) [UML – 12.3.4]

### A.5.5 IXIT

#### A.5.5.1 «TV_IXIT»

The stereotype «TV_IXIT» extends the metaclass Comment with multiplicity [0..1]. It is intended to capture the semantics of *IXIT* in the RM-ODP technology language.

#### A.5.5.2 Attributes

No tag definitions are defined for this stereotype

#### A.5.5.3 Constraints

No constraints are defined for this stereotype

#### A.5.5.4 Semantics

An «TV_IXIT» Comment is mapped to an *IXIT*.

#### A.5.5.5 Notation

UML standard syntax for Comment with stereotype is used.

#### A.5.5.6 References

RM-ODP: *IXIT* [Part 3 – 9.1.3]

UML: Comment (from Kernel) [UML – 7.3.9]

## A.6 Conformace profile

*Conformance* relates an *implementation* to a specification. Any *proposition* that is true in the specification must be true in its *implementation* (see [6.6]).

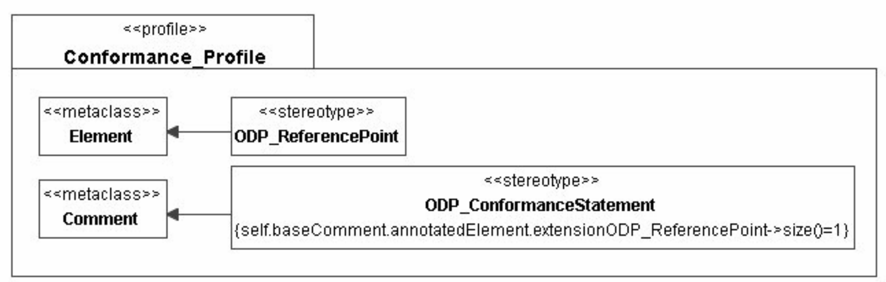



**Figure A.1 – UML 2.0 profile for conformance**

### A.6.1 Reference point

#### A.6.1.1 «ODP_ReferencePoint»

The stereotype «ODP_ReferencePoint» extends metaclass Element with multiplicity [0..1]. It is intended to capture the semantics of a *reference point* in RM-ODP.

#### A.6.1.2 Attributes

No tag definitions are defined for this stereotype.

#### A.6.1.3 Constraints

No constraints are defined for this stereotype.

#### A.6.1.4 Semantics

An «ODP_ReferencePoint» UML element is mapped to a *Reference point*. A *reference point* is a point at which *conformance* may be tested and which will, therefore, needs to be accessible for test.

#### A.6.1.5 Notation

UML standard syntax for Element with stereotype is used.

#### A.6.1.6 References

RM-ODP: *Reference point* [Part 2 – 10.6]

UML: Element (from Kernel) [UML – 7.3.14]

### A.6.2 Conformance statement

#### A.6.2.1 «ODP_ ConformanceStatement»

The stereotype «ODP_ConformanceStament» extends the metaclass Comment with multiplicity [0..1]. It is intended to capture the semantics of *conformance statement* in RM-ODP.

#### A.6.2.2 Attributes

No tag definitions are defined for this stereotype

#### A.6.2.3 Constraints

Every «ODP_ConformanceStament» comment should be attached to a *reference point*, that is, a UML element stereotyped «ODP_ReferencePoint».

#### A.6.2.4 Semantics

An «ODP_ConformanceStament» comment is mapped to a *conformance statement*. A *conformance statement* is a *statement* that identifies *conformance points* of a specification and the behaviour which must be satisfied at these points.

*Conformance statements* will be mapped to UML comments stereotyped «ODP_ConformanceStatement», attached to the UML model elements (stereotyped «ODP_ReferencePoint») that map to the corresponding *reference points*. These comments will describe the *conformance criteria* that should be satisfied at the *reference point*. Therefore, *conformance criteria* are those model elements stereotyped «ODP_ReferencePoint», which have also attached a «ODP_ConformanceStatement» comment. It is possible to attach multiple «ODP_ConformanceStatement» comments to one model element stereotyped «ODP_ReferencePoint», thus declaring several *conformance criteria* at the same *reference point*.

#### A.6.2.5 Notation

UML standard syntax for Comment with stereotype is used.

#### A.6.2.6 References

RM-ODP: *Conformance statement* [Part 2 – 15.1]

UML: Comment (from Kernel) [UML – 7.3.9]

## A.7 Structuring the specifications

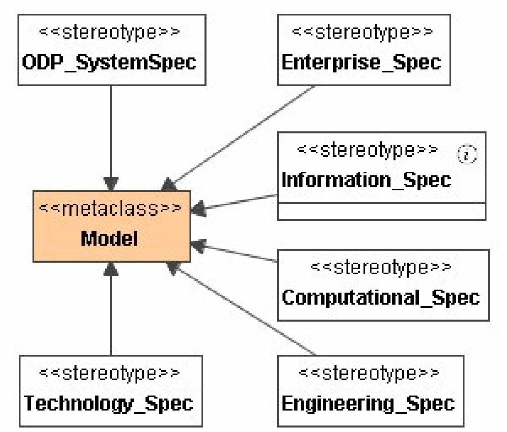



**Figure A.2 – Stereotypes for structuring the specifications**

### A.7.1 ODP system and viewpoint specifications

#### A.7.1.1 «ODP_SystemSpec», «Enterprise_Spec», «Information_Spec», «Computational_Spec», «Engineering_Spec», «Technology_Spec»

The stereotypes «ODP_SystemSpec», «Enterprise_Spec», «Information_Spec», «Computational_Spec». «Engineering_Spec» and «Technology_Spec »extend metaclass Model with multiplicity [0..1].

#### A.7.1.2 Attributes

No tag definitions are defined for these stereotypes.

#### A.7.1.3 Constraints

No constraints are defined for these stereotypes.

#### A.7.1.4 Semantics

The UML specifications of the ODP system will consist of one top-level UML model stereotyped «ODP_SystemSpec» that contains a set of UML models, one for each viewpoint specification, each stereotyped as «<X>_Spec», where <X> is the viewpoint concerned (see [6.5]).

#### A.7.1.5 Notation

UML standard syntax for Model with stereotype is used.

#### A.7.1.6 References

RM-ODP: Framework [Part 3 – 4]

UML: Model (from Models) [UML – 17.3.1]

# Annex B An example of ODP specifications using UML

(This annex forms an integral part of this Recommendation | International Standard)

The following example illustrates the results of use of UML for representing ODP system specifications.

This annex is not normative.

Temporary Note 1 – The example included in this document is a proposal for discussion purposes only. Its purpose is twofold: to assist with development and proving of the main text, and to act as tutorial material when the text is published.

Temporary Note 2 – It was agreed at the Brisbane meeting that, although one example will be finally included in this Annex, limited examples to illustrate or explore particular issues are also possible.

## B.1 The Templeman Library System

Temporary Note – The following specifications of the Library system are rather long because they are presented to the WG mainly for discussion purposes, and therefore they include many details that could be omitted in the final version. Besides, they also present many justifications on the representation mappings used.

### B.1.1 Introduction

This is an example of an ODP specification of a Library system, using UML. The example is about the computerized system that supports the operations of a University Library, in particular those related to the borrowing process of the Library items. The system should keep track of the items of the University Library, its borrowers, and their outstanding loans. The library system will be used by the library staff (librarian and assistants) to help them record loans, returns, etc. The borrowers will not interact directly with the library system.

NOTE – In the following, the *library system* (or the *system*, for short) will refer to the computerized system that supports the library operations, while the *library* will refer to the business itself, i.e., the environment of the *system*.

Instead of a general and abstract library, this example is based on the regulations that rule the borrowing process defined at the Templeman Library at the University of Kent at Canterbury, a library that has been previously used by different authors for illustrating some of the ODP concepts.

### B.1.2 Rules of operation of the Library

The basic rules that govern the borrowing process of that Library are as follows:

(1) Borrowing rights are given to all academic staff, and to postgraduate and undergraduate students of the University.

(2) Library books and periodicals can be borrowed.

(3) The librarian may temporarily withhold the circulation of Library items, or dispose them when they are no longer appropriate for loan.

(4) For requesting a loan, the borrower must hand the books or periodicals to a Library assistant.

(5) There are prescribed periods of loan and limits on the number of items allowed on loan to a borrower at any one time. These rules may vary from time to time, the Librarian being responsible for setting the chosen policy. Typical limits are detailed below:

- Undergraduates may borrow eight books. They may not borrow periodicals. Books may be borrowed for four weeks.
- Postgraduates may borrow 16 books or periodicals. Periodicals may be borrowed for one week. Books may be borrowed for one month.
- Teaching staff may borrow 24 books or periodicals. Periodicals may be borrowed for one week. Books may be borrowed for up to one year.

(6) Items borrowed must be returned by the due day and time which is specified when the item is borrowed.

(7) Borrowers who fail to return an item when it is due will become liable to a charge at the rates prescribed until the book or periodical is returned to the Library, and may have borrowing rights suspended.

(8) Borrowers returning items must hand them in to an assistant at the Main Loan Desk. Any charges due on overdue items must be paid at this time.

(9) Failure to pay charges may result in suspension by the Librarian of borrowing facilities.

In the following, we will refer to these rules as the “textual regulations” of the Library system. They will be the starting point for the ODP specifications below.

It is important to note that the textual regulations above leave many details of the system unspecified, such as when or how a borrower suspension is lifted by the librarian, or the precise information that needs to be kept in the system for each user and Library item. The specification process followed here will help uncover such missing details progressively, so the appropriate stakeholders of the system can determine them by making the corresponding decisions.

### B.1.3 Expressing the Library System Specification in UML

This Annex describes a specification of the different ODP viewpoints of such a system, using UML. For each of the viewpoints, this specification uses the corresponding languages defined in RM-ODP and, where appropriate, interprets the languages in terms of the UML notation.

The UML specifications of the ODP system will consist of one top-level UML model stereotyped «ODP_SystemSpec» composed of five UML models with the specifications of the five ODP viewpoints (Figure B.1). These models will be described in the following clauses.

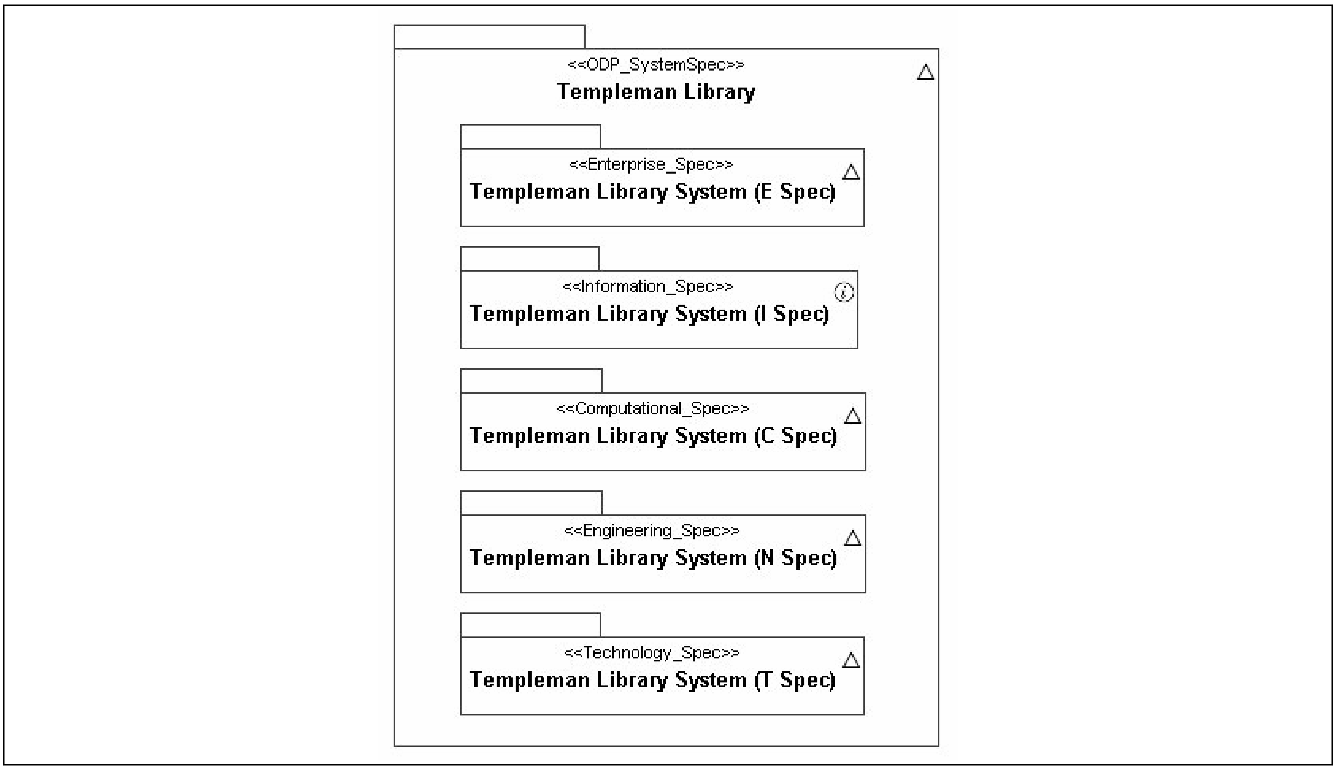



**Figure B.1 – UML specification of the ODP system**

## B.2 Enterprise specification in UML

### B.2.1 Basic enterprise concepts

The *enterprise viewpoint* is an abstraction of the system that focuses on the purpose (i.e., *objective*), *scope* and *policies* for that *system* and its *environment*. It describes the business requirements and how to meet them, but without having to worry about other system considerations, such as particular details of its software architecture, its computational processes, or the technology used to implement it.

Four key concepts of enterprise language are: *system*, *scope*, *enterprise specification*, and *field of application*. In the first place, the *system* to be specified is a computerized system that supports the operations of a University Library, in particular those related to the borrowing process of the Library items. This *system* has a name “The Templeman Library System” (or “TLS” for short).

The *scope* of the TLS system describes its expected *behaviour*, i.e., the way it is supposed to work and interact with its *environment* in the business context. In the enterprise language, the *scope* of the system is expressed as the set of roles it fulfils.

In UML, the enterprise specification of the TLS system is represented by one UML model, stereotyped «Enterprise_Spec», which is shown in Figure B.1.

NOTE – In the figures that follow, to improve the clarity of the diagrams, the icons shown in Figure B.2 have been used to represent instances of the corresponding stereotypes.

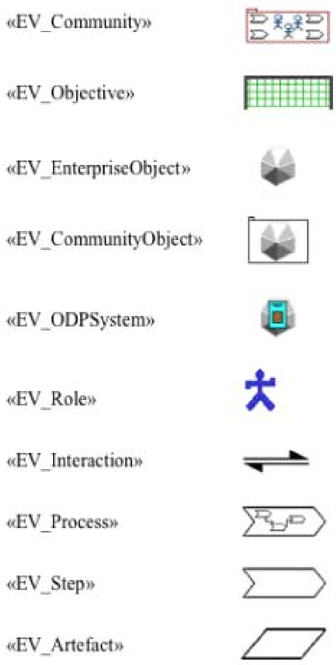



**Figure B.2 – Enterprise Language Icons**

The ODP Enterprise Language specification does not prescribe any particular method for building the enterprise specification of a system, as the approach taken will depend very much on the system being specified, the business that it will support, and the constraints that arise from the environment in which the system will operate. For this example, the following process has been followed:

1. Identify the *communities*, with which the system is involved, and their *objectives*.
2. Define the *behaviour* required to fulfil the *objectives* of the *communities*. This may be in the form of *processes*, their corresponding *actions*, and the participant *roles* in them. *Objects* may participate in *actions* as *actors*, *artefacts* (if they are just referenced in the *action*), and *resources* (*artefacts* essential to the *action* that may become unavailable or used up).
3. Alternatively, or at the same time, depending on the modelling objectives, behaviour may be expressed in the form of interactions between objects fulfilling roles.
4. Identify the *enterprise objects* in each *community*, and how they fill the *roles*.
5. Identify the *policies* that govern the *actions* (*permissions*, *obligations*, *authorizations*, *prohibitions*), and the effects of the possible violations of those *policies*.
6. Identify any behaviour that may change the structure or the members of each *community* during its lifetime, and the *policies* that govern such *behaviour*.

6 Identify any behaviours that may change the rules that govern the system, and the *policies* that govern such behaviours (changes in the structure, behaviour or *policies* of a *community* can occur only if the specification includes the behaviour that can cause those changes).

7 Identify the *actions* that involve *accountability* of the different *parties*, and the possible *delegations*.

Of course, the order of these activities needs not necessarily be linear, and nor will all activities be appropriate for all modelling situations.

### B.2.2 Communities

As shown in Figure B.2, the enterprise specification of the library example contains two *communities* (the **Library Community** and the **Academic Community**). Each of these is specified in a package, stereotyped as «EV_CommunityContract», containing a component, stereotyped as «EV_Community» (as well as other model elements specifying other aspects of the community), which has a dependency, stereotyped as «EV_RefinesAsCommunity», from a *community object* (**Library** and **Academic Community**) which maps to the *community object* that specifies the *community* when considered as a single *object*. (Note, the **Academic Community** is included only to illustrate the principle that, at the top level, there may be more than one community. The **Academic Community** is not further detailed in this example.)

The *field of application* of the enterprise specification describes the properties that the environment of the ODP system must have for such specification to be used. It is described in a comment, stereotyped as «EV_FieldOfApplication» attached to the top level UML model, stereotyped «EV_FieldOfApplication», containing the enterprise specification of the system (see Figure B.2).

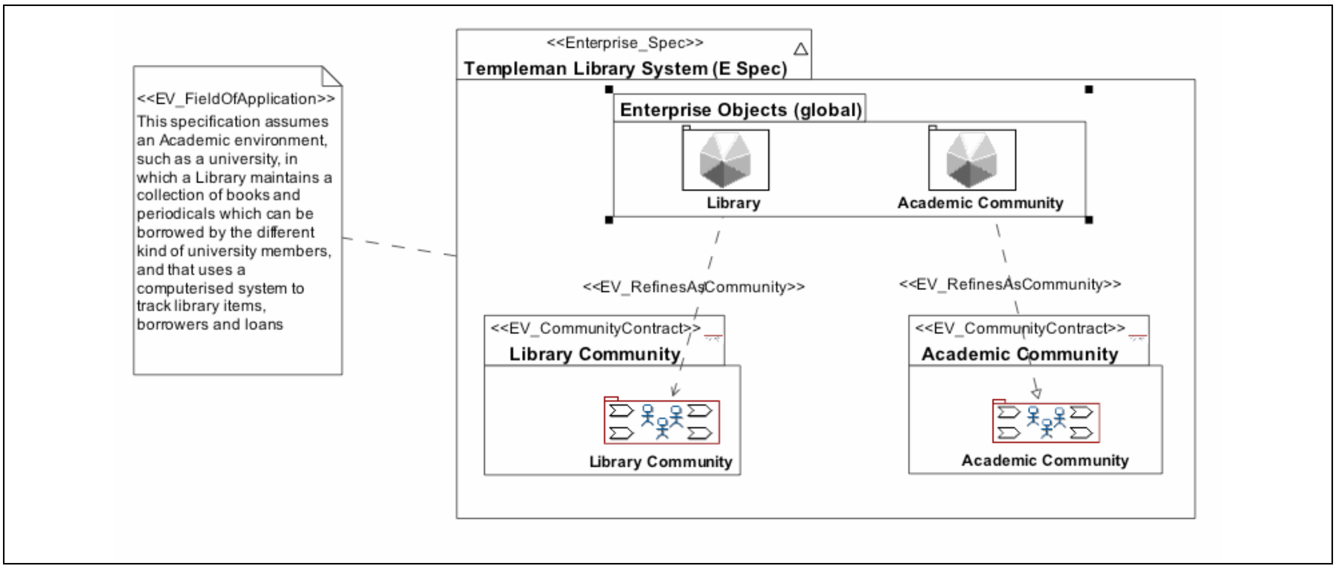



**Figure B.3 – UML Enterprise specification of the Library System**

A *community* is a configuration of *objects* modelling a collection of entities (e.g. human beings, information processing systems, resources of various kinds, and collections of these) that are subject to some implicit or explicit *contract* governing their collective behaviour, and that has been formed for a particular *objective*.

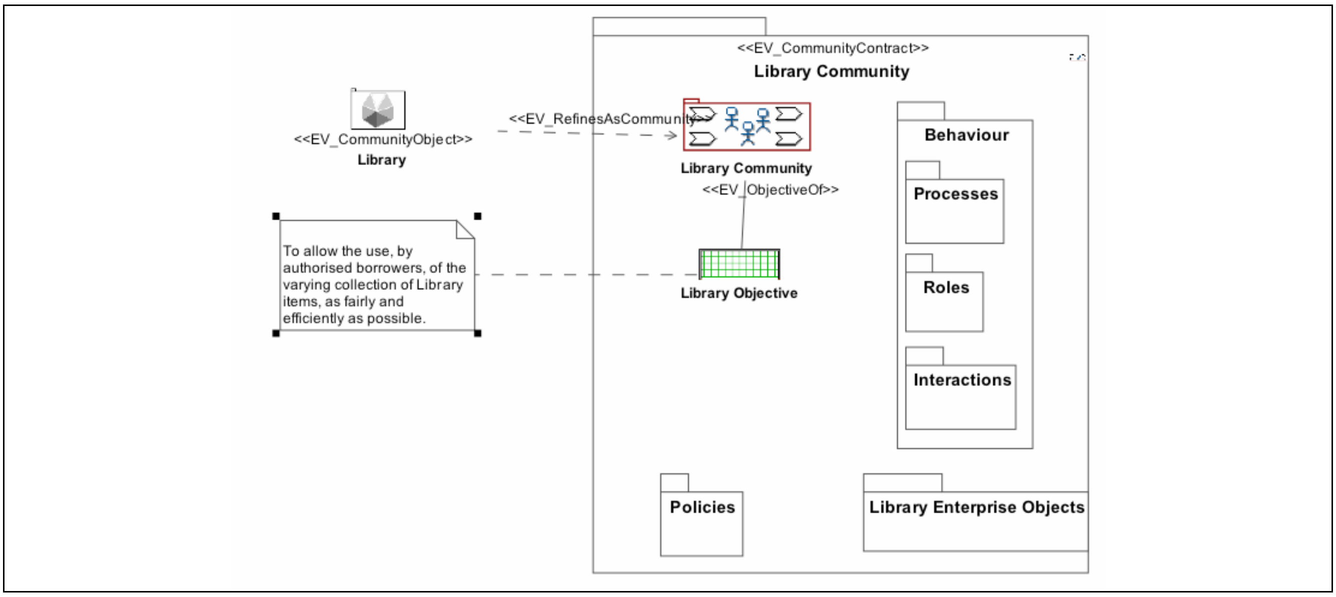



**Figure B.4 – UML specification of the Library system community**

The UML package representing the *community* is stereotyped «EV_Community», and contains three UML packages with the description of the structure, *behaviour*, and *policies* for the *community*, respectively, and one class (stereotyped «EV_Objective») with the description of the *community objective*. That class (called **Library Objective**) describes the *community objective* as follows: "To allow the use, by authorised borrowers, of the varying collection of Library items, as fairly and efficiently as possible".

The structure of the *community* is described in the **Enterprise Objects** package; see B.2.5.

### B.2.3 Behaviour

The **Behaviour** package describes the behaviour required of the *community* to meet its objective. There are three packages, **Roles**, **Process** and **Interactions**.

- The **Roles** package contains the classes representing the Library *roles* (see Figures B.5 and B.6).
- The **Processes** package contains the specification of the *community processes*. In general, the *processes* of a *community* can be organized into sub-packages attending to their particular characteristics: business processes, administrative processes, security processes, etc. In this example, for simplicity, the **Processes** package contains three sub-packages, **Borrowing Processes**, **Fining Processes** and **Administrative Processes**, which describe all the Library *processes* (see Figure B.7).
- The **Interactions** package specifies the *interaction types* of the *community interactions* among their *roles* (Figure B.10).

### B.2.4 Processes

*Processes* describe behaviour in terms of (partially ordered) sets of *steps*, which are related to achieving some particular sub-*objective* within the *community*. *Steps* are abstractions of *actions*, which may hide some of the *objects* participating in the *actions*.

In general, the *processes* of a *community* can be organized into sub-packages attending to their particular characteristics: business processes, administrative processes, security processes, etc. In this example, for simplicity, the **Processes** package contains three sub-packages, namely **Borrowing Processes**, **Fining Processes** and **Administrative Processes**, which describe all the **Library** *processes* (Figure B.5).

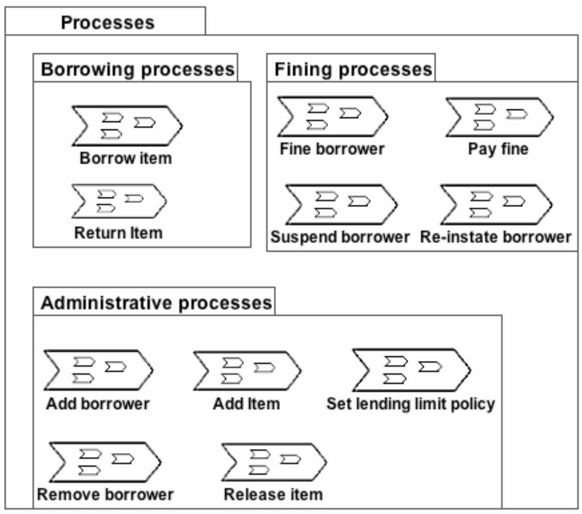



**Figure B.5 – Processes package structure**

Each *process* is represented by a UML activity, stereotyped «EV_Process», as shown in Figure B.4. Each of these activities has associated with it an Activity Diagram that describes the *steps* of the *process*, and the *roles* involved in these *steps* (either as *actor* or as *artefact roles*). *Actor roles* are mapped to the Activity Partitions (stereotyped «EV_Role»), and *artefact roles* are mapped to ObjectNodes (stereotyped «EV_Artefact»). For instance, Figure B.6 shows the Activity Diagram that specifies the **Borrow Item** *process*.

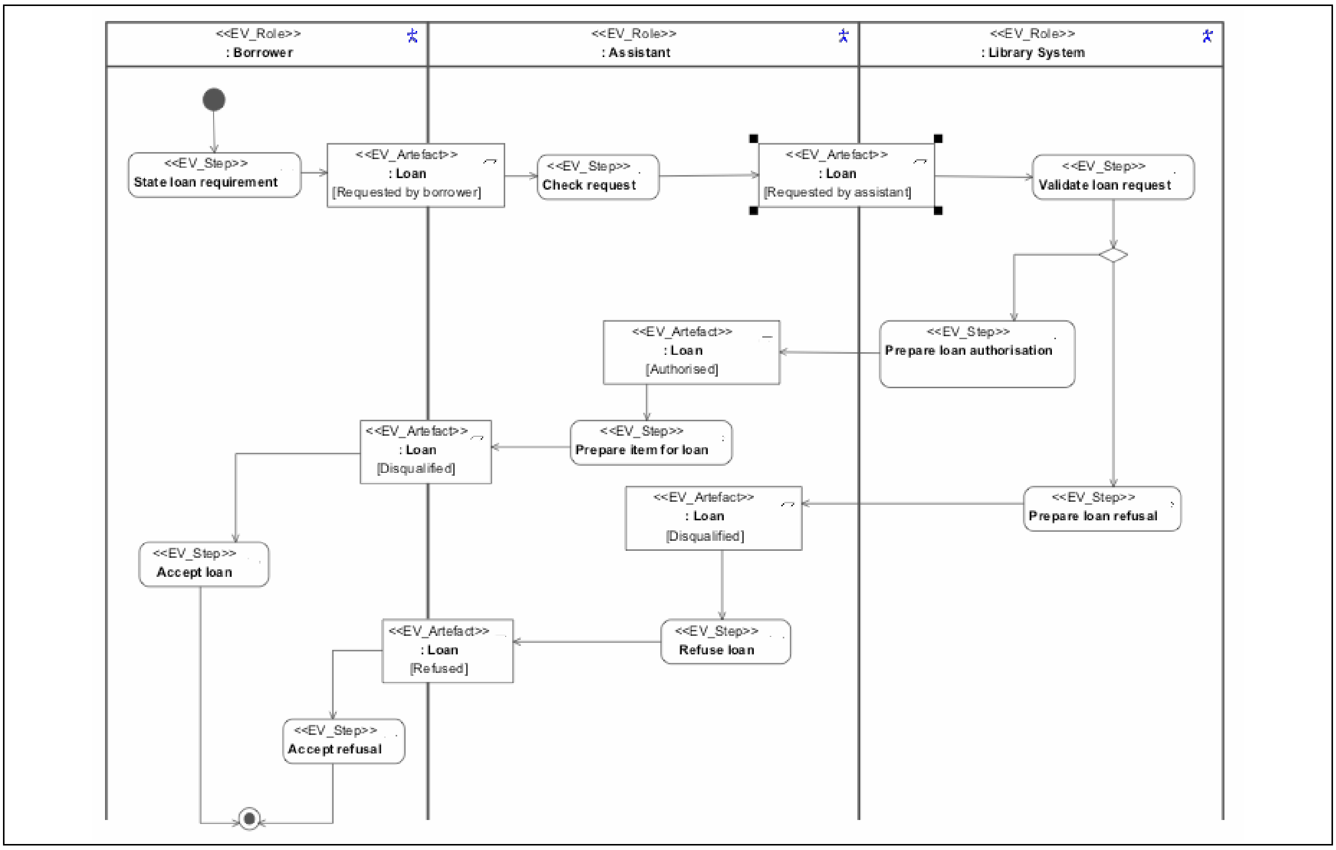



**Figure B.6 – Borrow item *Process***

### B.2.3 Roles

From the textual description of the Library (and, in real life more importantly, from discussions, interviews and workshops with stakeholders) we can identify several *roles* in the **Library** *community*, in particular borrowers with various privileges, librarians, library assistants, and the computerized system that supports the Library operations (**Library System**). Figure B.5 shows these Library roles within the package that specifies the *community*, each with a realization link to the component that maps to the *community*.

An *community* may also be expressed as a composite *object* when considered at a more abstract level of detail and, similarly, any enterprise *object* may itself be refined as a *community* at a more detailed level. Thus, there is also a **Library** *community object* that represents the community when considered as a composite *object* and this has a dependency, stereotyped as «EV_RefinesAsCommunit», onto the component that maps to the *community*.

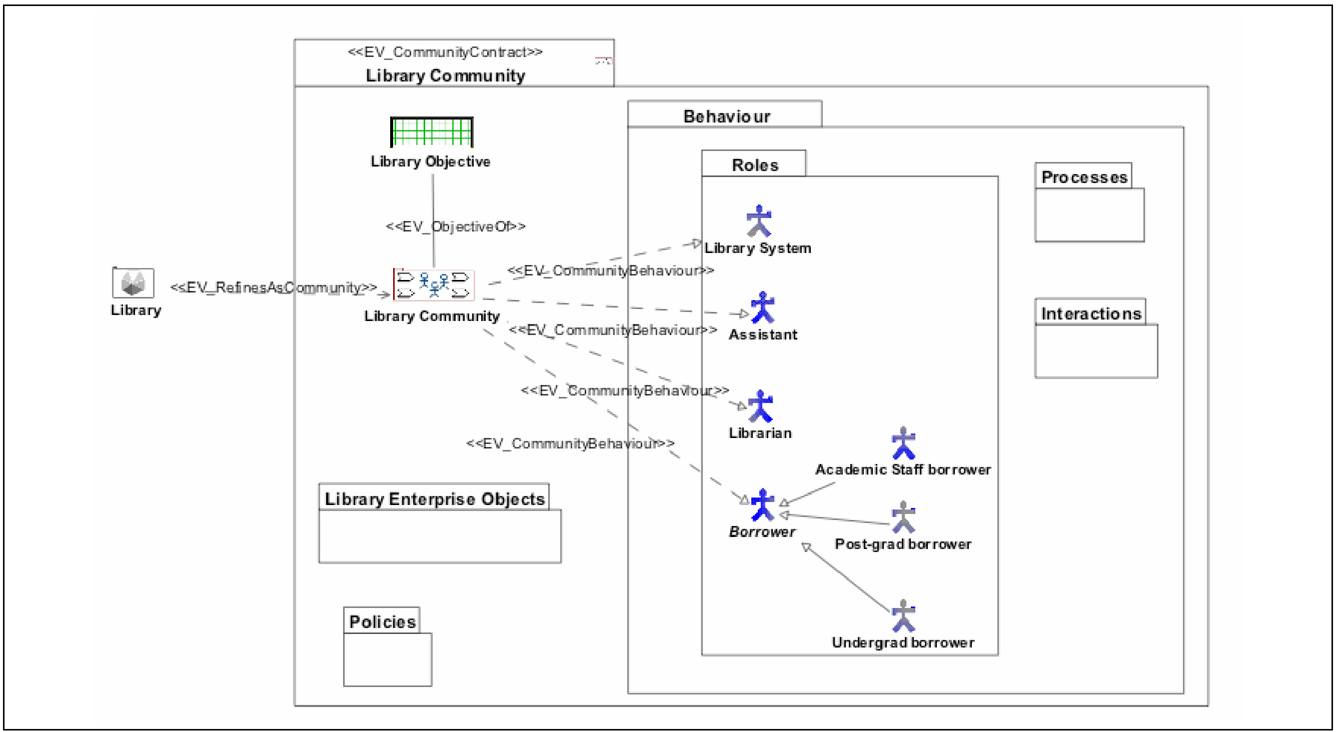



**Figure B.7 – Structure of the Library community described by the specification of its enterprise roles**

### B.2.4 Enterprise Objects

*Roles* are fulfilled by *enterprise objects*. The fulfilment of *actor roles* in a *community* by *enterprise objects* is governed by *assignment rules*. Using UML, fact that an *actor role* may be fulfilled by an *enterprise object* is expressed by an association, stereotyped as «EV_FulfilsRole», between the classes that express the *objects* and the *roles* concerned. *Assignment rules* can be constrained by the *policies* of the system, in which case there would be model links between the *roles* and model elements expressing the *policies*. Figure B.8 shows the UML expression of the basic (i.e. unconstrained by any *policies*) assignment *rules* of the **Library** *community*.

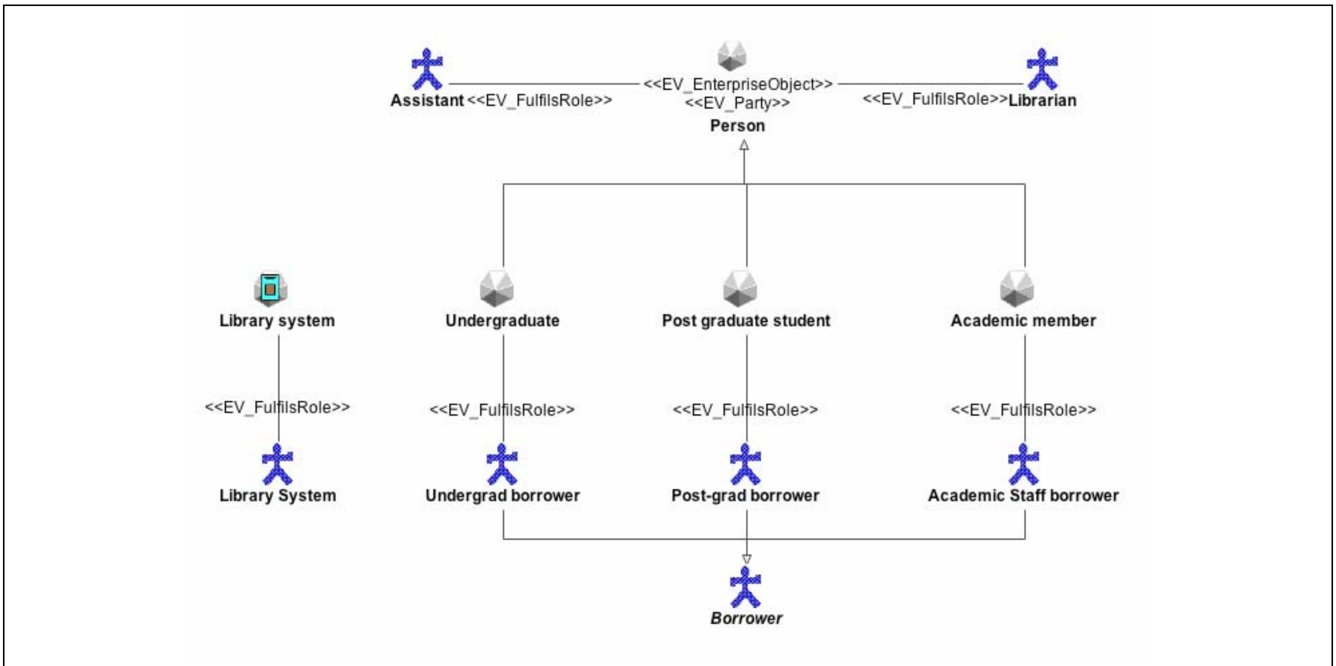



**Figure B.8 – Actor Role fulfilment and assignment rules**

*Enterprise objects* may also participate in *actions* by fulfilling *artefact roles*. In this example, **Loans** are *enterprise objects* that represent the relationship that is established between a borrower and an item that is established when she requests the item, and continues for a period from either the loan being refused or the item, having been loaned, being returned. **Loans** fulfil *artefact roles* in several *actions* (from *interaction* model, see B2.5), as shown in Figure B.9.

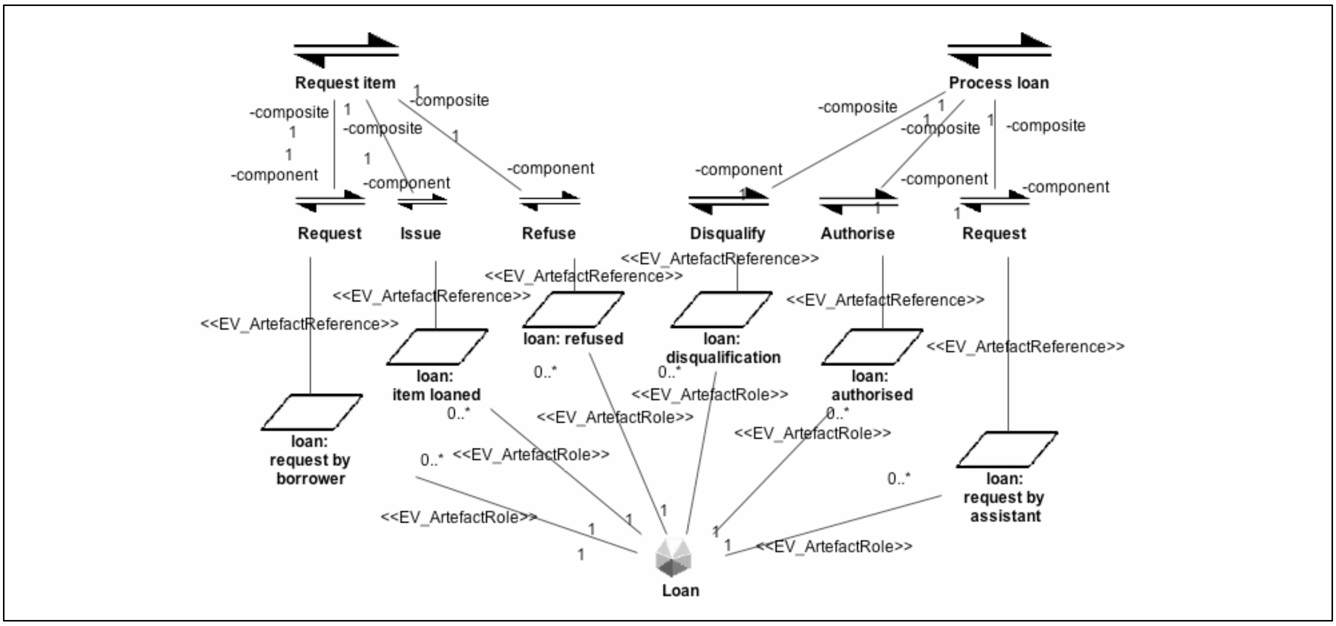



**Figure B.9 – Loan as an Artefact**

In summary, the *enterprise objects*, and the relationships between them, that have *roles* (either *actor* or *artefact*) in the **Library** *community* are shown in Figure B.10.

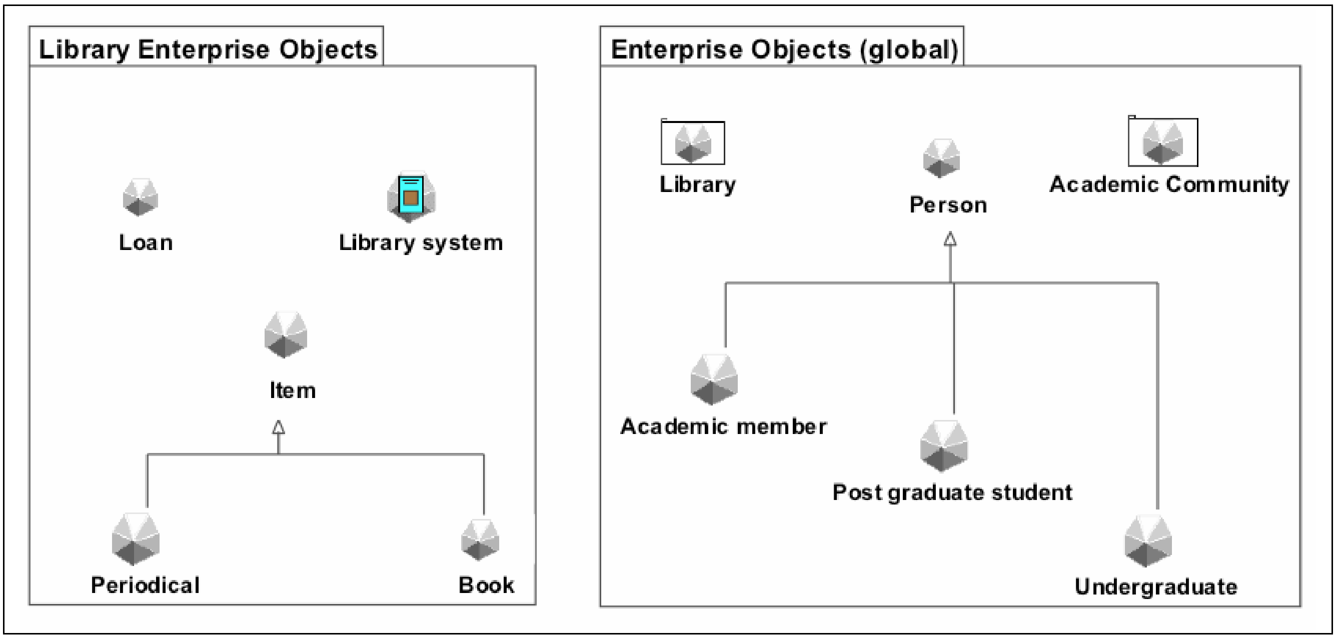



**Figure B.10 – Enterprise objects**

### B.2.5 Interactions

*Behaviour* can also be expressed in terms of *interactions* between *roles* in a *community*. Thus, using the *roles* described for the **Library** *community* above, we can now specify the *behaviour* of the *enterprise objects* of the *community*, assigning *actions* to one or more *roles*.

*Interactions* are described in the **Interactions** package, which is organized into sub-packages reflecting the basic purpose behind each interaction set, and therefore structured similarly to the **Processes** package (See Figure B.5). This is shown in Figure B.11. The contents of one of these sub-packages are also shown, and we can see that there are two *interactions* associated with **Borrow Item**: **Request Item** and **Process Loan**.

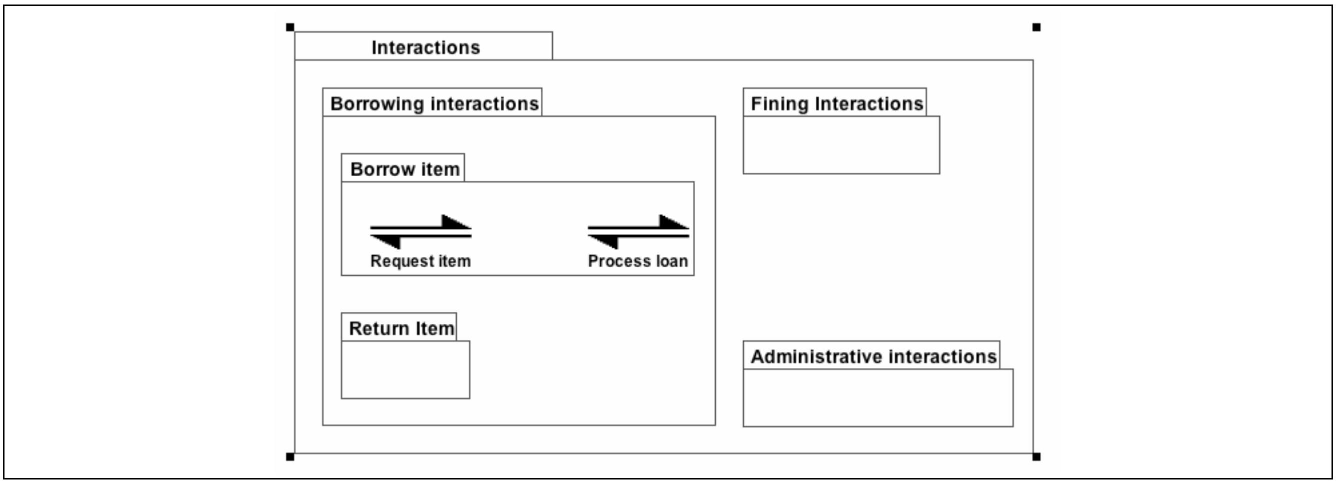



**Figure B.11 – Interactions package structure**

The relationships between the classes expressing the *interactions* involved in the *behaviour* of requesting a loan, and those classes expressing the *roles* involved in such *interactions* is shown in Figure B.12. There are two *interactions* in this case: **Request Item** in which **Borrower** and **Assistant** are involved, and **Process Loan** in which **Assistant** and **Library System** are involved. In each case the relationship is expressed with an association, stereotyped as «EV_InteractionInitiator» or «EVInteractionResponder» as appropriate.

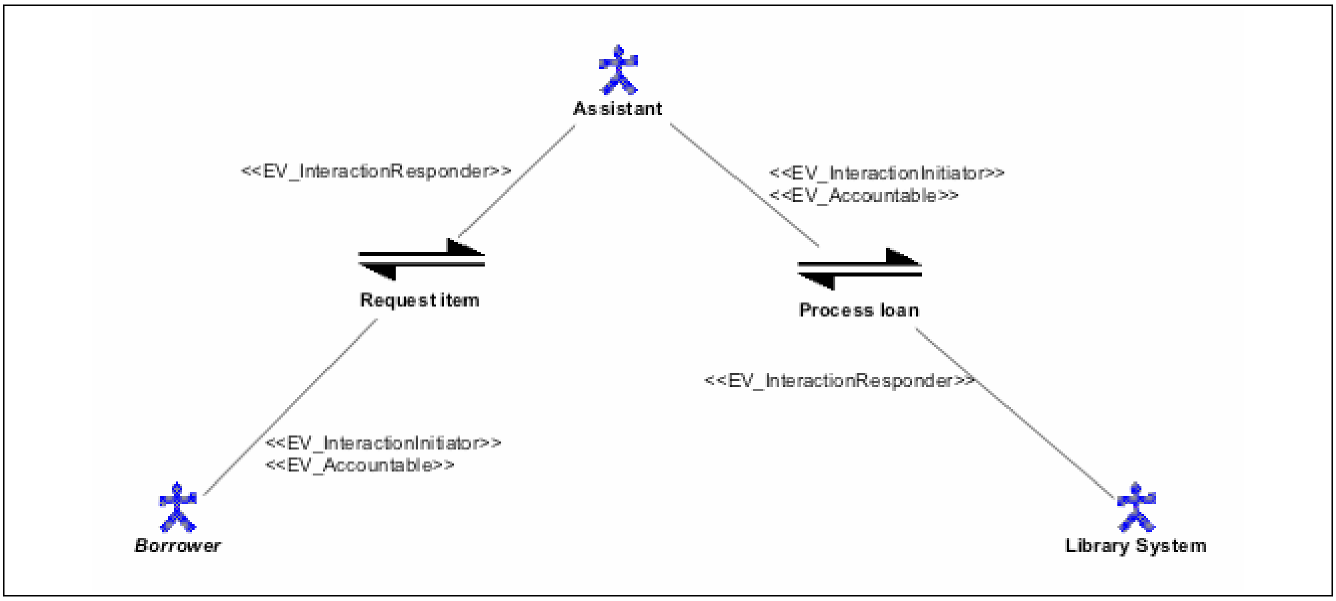



**Figure B.12 – Roles and behaviour – class diagram for Borrow Item behaviour**

*Interactions* can be refined into sets of more detailed *interactions*. Thus, Figure B.13 shows **Request Item** (an interaction initiated by the object fulfilling the *role* **Borrower** and responded to by the object fulfilling the *role* **Assistant)** as composed of three *interactions*, **Request**, **Issue** and **Refuse** each of which has an association, stereotyped as «EV_ArtefactReference» with an *artefact role*, **loan: request by borrower**, **loan: item loaned** and **loan: refused** respectively. The *artefact roles* are expressed by signals each stereotyped as «EV_Artefact», and are defined as fulfilled by the *enterprise object*, **Loan**.

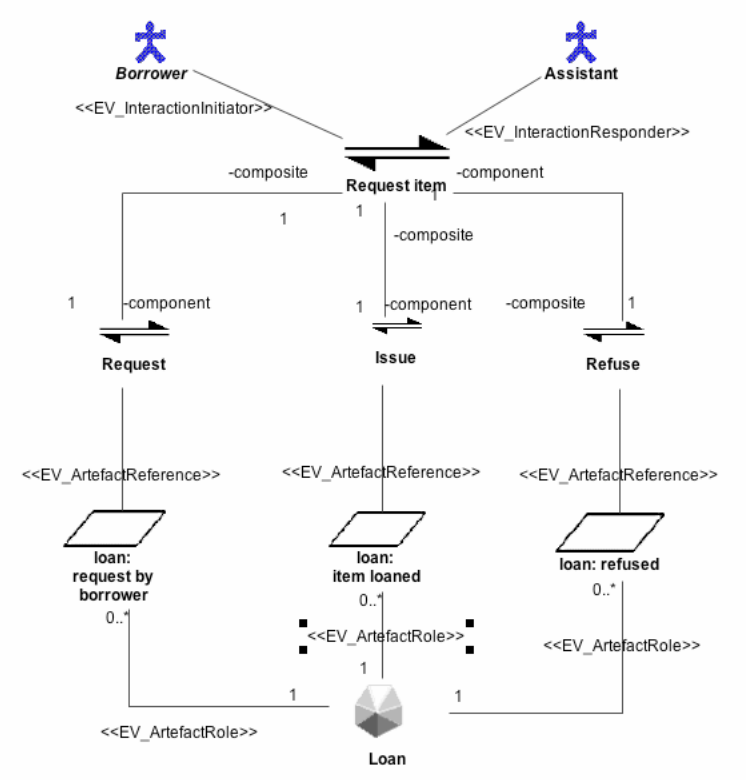



**Figure B.13 – Roles and behaviour: class diagram for Request Item**

Similarly, Figure B.14 defines **Process Loan** (initiated by the *role* **Assistant** and responded to by the *role* **Library System**) as an *interaction* composed of three *interactions*, **Request**, **Authorize** and **Disqualify.** Each of these has an association, stereotyped as «EV_ArtefactReference» to a signal, stereotyped as «EV_Artefact», expressing an *artefact role* of the **Loan** enterprise object: **loan: request by assistant**, **loan: authorised** and **loan: disqualification** respectively.

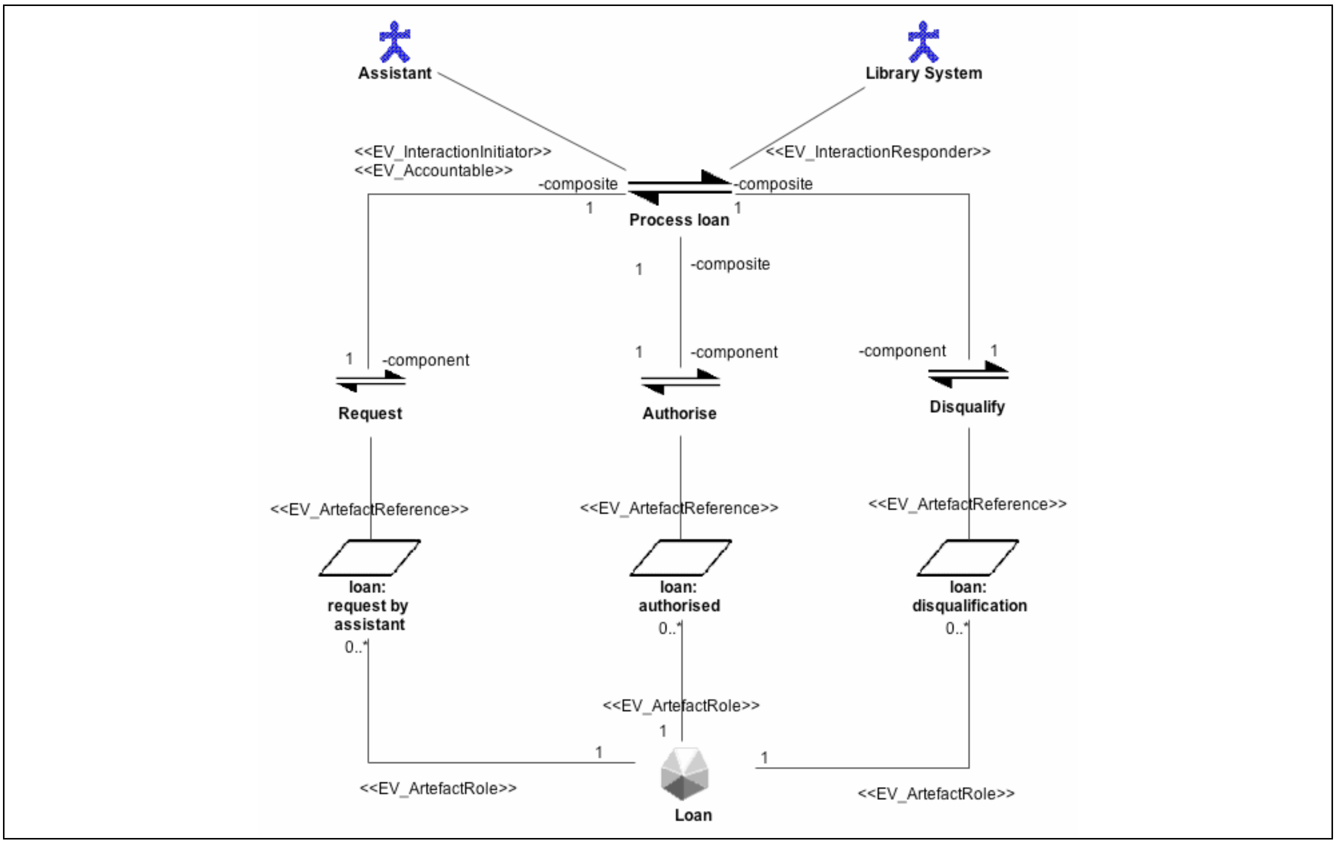



**Figure B.14 – Roles and behaviour – class diagram for Process Loan**

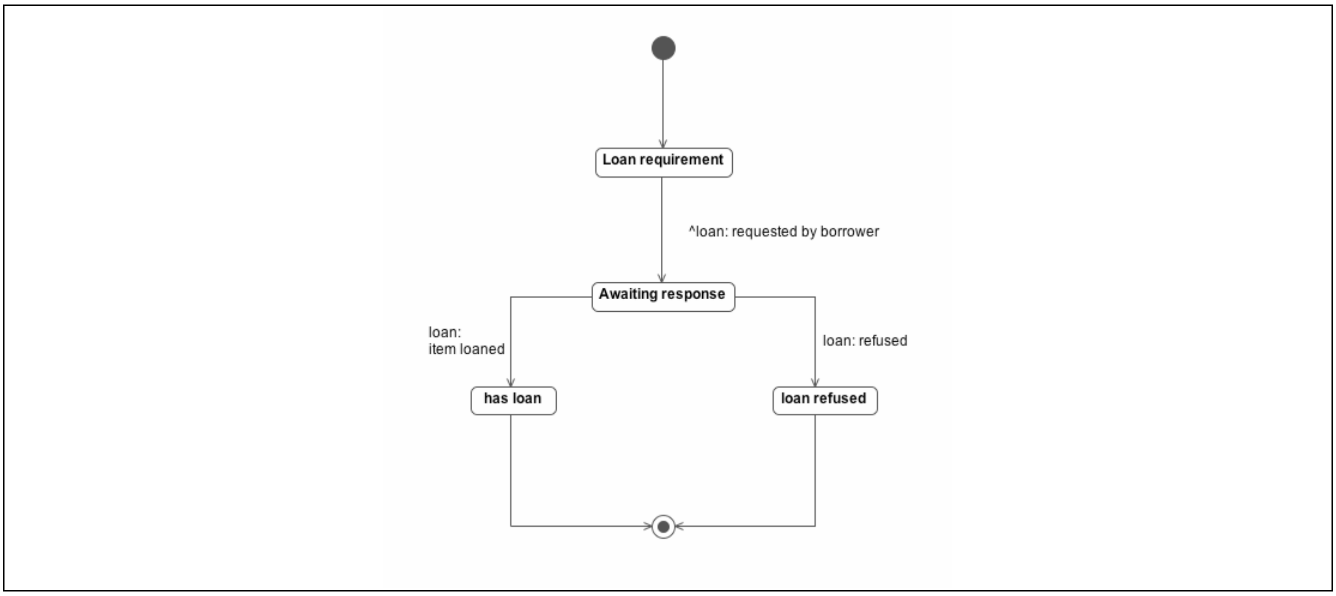



**Figure B.15 – Roles and behaviour – state diagram for Borrower role**

Figure B.15 shows the state machine for the *behaviour* defined for the *role* of **Borrower**. In the **Loan requirement** state a **Loan** in the **Requested by borrower** state is generated and there is a transition to the **Awaiting response** state with the sending of a **loan: requested by borrower** signal, expressing an *artefact role* of the *enterprise object* **Loan** in the **Request** *interaction* in the **Request Item** *interaction*. There is a transition from the **Awaiting response** state on the receipt of either a **loan: item loaned** signal or a **loan: refused** signal. The **loan: item loaned** signal expresses an *artefact role* of the *enterprise object* **Loan** in the **Issue** *interaction* in the **Request Item** *interaction*; the **loan: refused** signal expresses an *artefact role* of the *enterprise object* **Loan** in the **Refuse** *interaction* in the **Request Item** *interaction*. There is an automatic transition from either the has **loan state** or the **loan refused** state to the **final state.**

Figure B.16 shows the state machine for the *behaviour* defined for the *role* of **Assistant**. On receipt of a **loan: request by borrower** signal, expressing an *artefact role* of the *enterprise object* **Loan** in the **Request** *interaction* in the **Request Item** *interaction*, there is a transition from the starting state to the **initiating loan processing** state. In this state a **Loan** in the **Requested by assistant** state is generated and there is a transition to the **awaiting response** state with the sending of a **loan: requested by assistant** signal expressing an *artefact role* of the *enterprise object* **Loan** in the **Request** *interaction* in the **Process Loan** *interaction*. There is a transition from the **awaiting response** state on the receipt of either a **loan: authorised** signal or a **loan: disqualification** signal. The **loan: authorised** signal expresses an *artefact role* of the *enterprise object* **Loan** in the **Authorize** *interaction* in the **Process Loan** *interaction*; the **loan: disqualification** signal expresses an *artefact role* of the *enterprise object* **Loan** in the **Disqualify** *interaction* in the **Process Loan** *interaction*. In the **preparing item for loan** state the item is prepared for loan and there is a transition to *behaviour* completion with the sending of a **loan: item loaned** signal expressing an *artefact role* of the *enterprise object* **Loan** in the **Issue** *interaction* in the **Request Item** *interaction*; in the **preparing loan refusal** state a refusal is prepared and there is a transition to *behaviour* completion with the sending of a **loan: refused** signal expressing an *artefact role* of the *enterprise object* **Loan** in the **Refuse** *interaction* in the **Request Item** *interaction*.

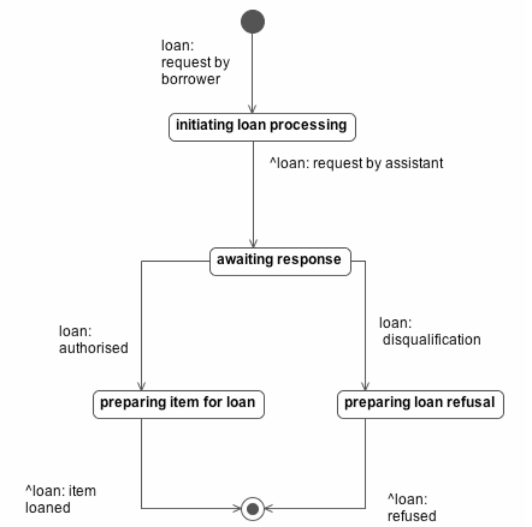



**Figure B.16 – Roles and behaviour – state diagram for Assistant role**

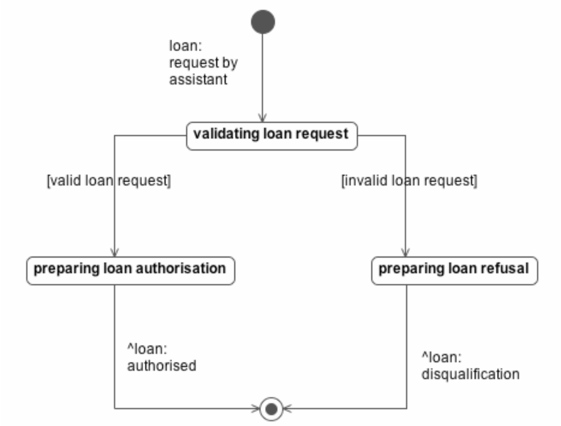



**Figure B.17 – Roles and behaviour – state diagram for Library System role**

Figure B.17 shows the state machine for the behaviour defined for the *role* of **Library System**. On receipt of a **loan: request by assistant** signal expressing an *artefact role* of the *enterprise object* **Loan** in the **Request** *interaction* in the **Process Loan** interaction, there is a transition from the starting state to the **validating loan request** state. In the **validating loan request** state the system decides whether the loan request is valid or invalid. If the loan request is valid there is a transition to the **preparing loan authorization** state where the authorization is prepared and there is a transition to *behaviour* completion with the sending of a **loan: authorised** signal expressing an *artefact role* of the *enterprise object* **Loan** in the **Authorize** *interaction* in the **Process Loan** *interaction*. If the loan request is invalid there is a transition to the **preparing loan refusal** state where a refusal is prepared and there is a transition to *behaviour* completion with the sending of a **loan: disqualification** signal expressing an *artefact role* of the *enterprise object* **Loan** in the **Disqualify** *interaction* in the **Process Loan** *interaction.*

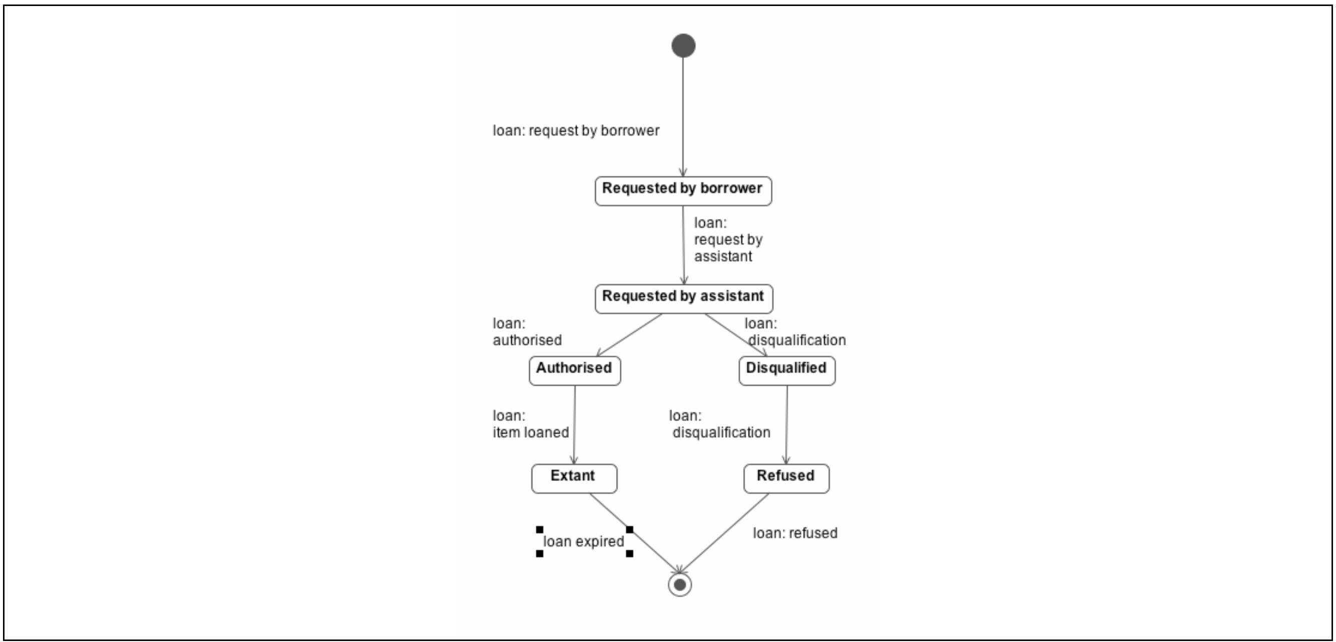



**Figure B.18 – Roles and behaviour – States of the Loan enterprise object**

Figure B.18 defines the top-level state machine for the **Loan** *enterprise object*.

The transition from the initial state to the **Requested by borrower** state represents the creation of the **Loan** *enterprise object* as a result of an event that is the receipt of a **loan: request by borrower** signal: this signal expresses an *artefact role* of the *enterprise object* **Loan** in the **Request Item** *interaction*, and corresponds to the **Request** *interaction* in the **Request Item** *interaction* (see Figure B.13).

There is a transition from the **Requested by borrower** state to the **Requested by assistant** state as a result of an event that is the receipt of the **loan: request** signal; this signal expresses an *artefact role* of the *enterprise object* **Loan** in the **Process Loan** *interaction*, and corresponds to the **Request** *interaction* in the **Process Loan** *interaction* (see Figure B.14).

There is a transition from the **Requested by assistant** state to the **Authorized** state as a result of an event that is the receipt of the **loan: authorised** signal; this signal expresses an *artefact role* of the *enterprise object* **Loan** in the **Process Loan** *interaction*, and corresponds to the **Authorize** *interaction* in the **Process Loan** *interaction* (see Figure B.14).

There is a transition from the **Requested by assistant** state to the **Disqualified** state as a result of an event that is the receipt of the **loan: disqualification** signal; this signal expresses an *artefact role* of the *enterprise object* **Loan** in the **Process Loan** *interaction*, and corresponds to the **Disqualify** *interaction* in the **Process Loan** *interaction* (see Figure B.14).

There is a transition from the **Authorized** state to a composite state, **Loan extant,** as a result of an event that is the receipt of the **loan: item loaned** signal; this signal expresses an *artefact role* of the *enterprise object* **Loan** in the **Request Item** *interaction*, and corresponds to the **Issue** *interaction* in the **Request Item** *interaction* (see Figure B.13). The **Loan extant** composite state represents the *behaviour* of the **Loan** *enterprise object* during the period of the **Loan**.

There is a transition from the **Disqualified** state to the **Refused** state as a result of an event that is the receipt of the **loan: disqualification** signal. This signal expresses an *artefact role* of the *enterprise object* **Loan** in the **Request Item** *interaction*, and corresponds to the **Refuse** *interaction* in the **Request Item** *interaction* (see Figure B.13).

The transitions from the **Loan extant** composite state and the **Disqualified** state occur after a pre-determined time (set by an appropriate *policy*) and represent the termination of the **Loan** *enterprise object.*

### B.2.7 Policies

#### B.2.7.1 General

In an enterprise specification the concept, *policy*, is intended to be used where the desired *behaviour* of the system may be changed to meet particular circumstances.

The **Policies** package specifies the *community policies*, which constrain the structure and/or the *behaviour* of the *community*. Therefore, the elements of that package will constrain the elements of the other two UML packages in the **Library Community** package (**Behaviour** and **Library Enterprise Objects**).

Providing an independent and modular specification of *policies* will enable the definition and implementation of some traceability mechanisms, both intra- and inter-viewpoints. Within the UML expression of the enterprise specification of a system, we need to be able to list all the model elements affected by a given *policy*, and all the *policies* that constrain a given model element, in case there is a change in the specification's elements or *policies*. But such independent expression of enterprise *policies* may also allow the definition of *correspondences* between these *policies* and other related elements from different ODP viewpoints (such as information *invariant schemata*). We expect UML modelling tools to exploit such traceability mechanisms, checking for absences of *policies* for some of the modelling elements, and also for *policy* conflicts and inconsistencies at various levels

In this relatively simple example, the aspects of the system that are most appropriate for use of this concept is in the rules regarding borrowing permissions (see B.1.2 rule (5)).

According to the considerations above, in order to be properly specified, *policies* need to identify the relevant enterprise elements to which they apply: *roles*, *objects*, *actions*, *processes*, *communities*, as well as their relationships. Such elements are precisely those described in the two other UML packages that form part of the enterprise specification of the system: **Enterprise Objects** and **Behaviour**.

#### B.2.7.2 Representing ODP policies in UML

In this example we will represent *policies* using the pattern shown in Clause 7.1.3, Figure 10, which corresponds to the elements that comprise the specification of an enterprise *policy* in the Enterprise Language [E/L – 7.9.2]:

- **description**: text with the description of the *policy* in natural language;
- **controllingAuthority**: an authority that controls the *policy* (in this case, a *role*);
- **relatedBehaviour**: an identified behaviour (i.e., *role*) that is subject to that authority;
- **relatedObjects**: optionally, an *object* or *objects* that may fulfil the *roles* involved;
- **specificationConstraint**: set of constraints on the modelling elements involved.
- **affectedBehaviour**: the subset of the related behaviour that is required, permitted, forbidden, or authorized.

The *behaviours*, *roles* and *objects* related to a *policy* specification in UML refer, of course, to the UML elements representing these *behaviours*, *roles* and *objects*, respectively. Such elements will normally be used as contexts in the constraints that specify the *policy*. Note that all *policy* statements are made in a context, that defines the elements in the specification to which the *policy* applies, and have a condition that specifies when the *policy* can be used. In this sense, OCL can be of real help. Each OCL constraint is expressed in a particular context, related to some element in the UML model. OCL statements can be directly associated to some model elements in a diagram, establishing an implicit context by attachment, or they can form part of a separate piece of specification in which the context of each statement is explicitly established by naming. Rules are represented as Constraints, expressed in a given notation (such as "OCL", or a specific policy language).

#### B.2.7.3 Expressing Loan policies in the Templeman Library

Figure B-19 shows the structure of the **Policies** package.

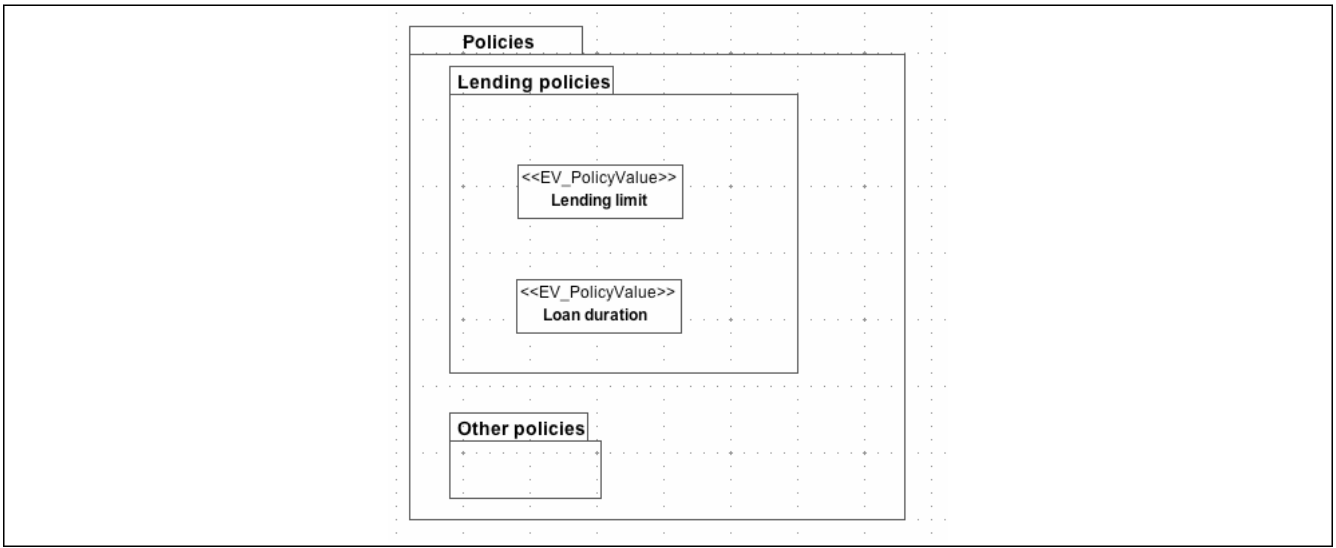



**Figure B.19 – Structure of the Policies package**

Details of the **Lending Policies** are shown in Figure B-20, which for illustrative purposes offers both behavioural modelling styles (i.e. with *processes* and *interactions*). From this it can be seen that the **Lending Limit Policy** is set by a *process* **Set lending limit policy** (located in the **Administrative Processes** package), and impacts on the *role* **Library System**, when taking part in the *process* **Borrow Item**, or the *interaction* of the same name.

Similarly the **Loan Duration policy** is set by the *interaction* **Set loan duration policy** (located in the **Administrative Interactions** package), and impacts on the *role* **Library System**, when taking part in the *process* **Fine Borrower**, or the *interaction* of the same name.

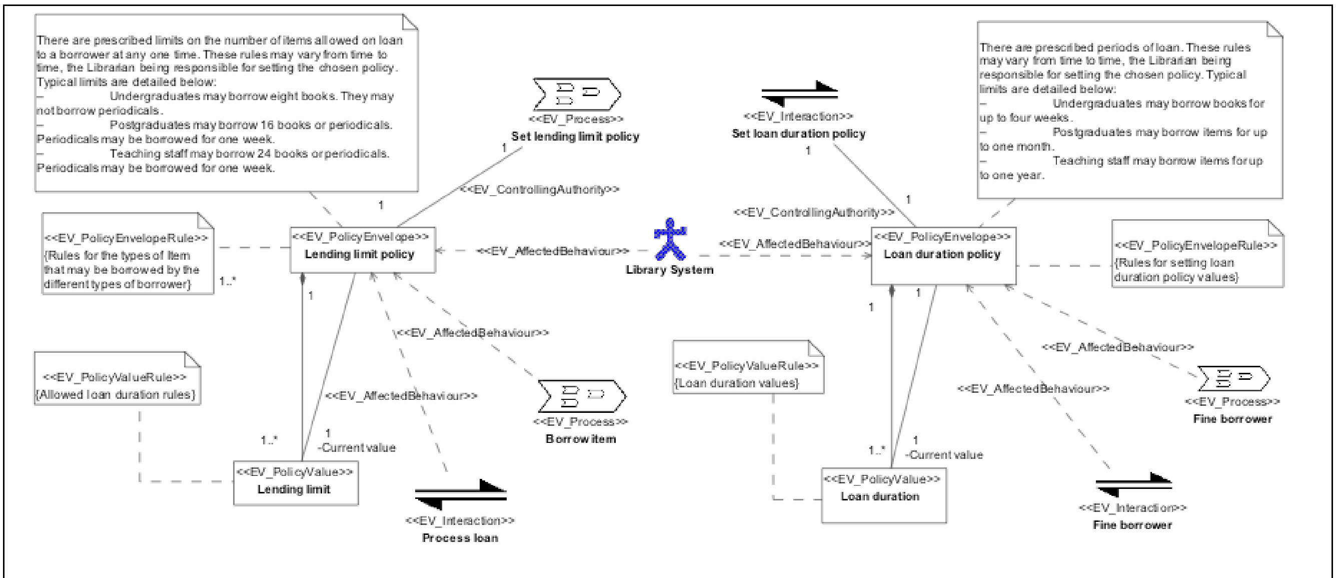



**Figure B.20 – Examples of policy expressions**

### B.2.8 Accountability

An enterprise specification should also identify those *actions* that involve *accountability* of a *party*, where a *party* represents a natural person or any other entity considered to have some of the rights, powers and duties of a natural person. *Principal parties* are responsible for the *acts* of any *parties* acting as their delegated *agents*, including their possible *commitments*, *prescriptions*, *evaluations*, *declarations*, and further *delegations*.

*Accountable parties* in a given *process* or *action* are represented in the UML diagram that defines such *process* or *action*. Stereotype «EV_Accountable» in an association between an *actor* and an *action* indicates the *actor* that is *accountable* for the *action*. Figure B.21 shows an example of the use of such a stereotype, indicating that the **Borrower** is the *accountable* party for the **Request Item** *action*, and the **Assistant** is *accountable* for the **Process Loan** *action*.

*Delegations* are represented in UML by associations between *roles* in activity diagrams stereotyped «EV_Delegation», showing the *principal* and *agent parties* of each *delegation*. Such associations allow delegated *parties* to initiate or participate in *actions* on behalf of their *principals*. In particular, Figure B.22 specifies that the **Assistant** can delegate his *actions* to a **Librarian** (i.e., the **Librarian** can act as an **Assistant** in these cases). As previously mentioned, the *delegation* may convey some information about its duration, conditions, further *delegations* allowed, etc. Attributes of the «EV_Delegation» stereotype may be used to represent such kind information.

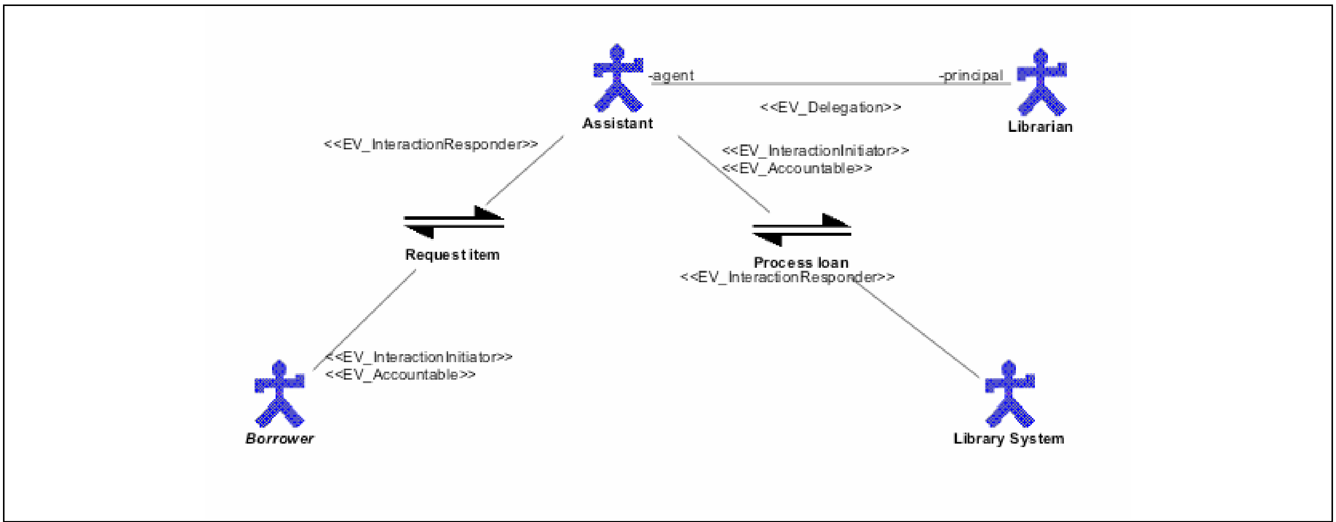



**Figure B.21 – Example of delegation**

### B.2.9 Interactions between Communities

Temporary Note: Text to illustrate the modelling of interactions between communities will be developed for the next draft of this Annex. This will illustrate the following scenarios:

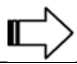
- an interaction between the Library System and an Academic System to update information on members of the University;
- some interaction of a member of the Library staff with the Academic Community that involves the Library staff member interacting with the Library System.

## B.3 Information specification in UML

### B.3.1 Overview

The information viewpoint is concerned with information modelling. An information specification defines the semantics of information and the semantics of information processing in an ODP system, without having to worry about other system considerations, such as particular details of its implementation, the computational process, or the nature of the distributed architecture to be used. The information specification in this clause defines both the basic concepts for information used in this specification, and the *invariant*, *static* and *dynamic schemata*.

NOTE – In the figures that follow, to improve the clarity of the diagrams, the following icons have been used to represent the corresponding stereotypes:

| | |
|---|---|
| «Information_Spec» |  |
| «IV_Object» |  |
| «IV_ObjectType» |  |
| «IV_ActionType» |  |
| «IV_InvariantSchema» |  |
| « IV_StaticSchema» |  |
| « IV_DynamicSchema» |  |

According to [8.4], the UML information specification of the Library system is given by one UML model, stereotyped «Information_Spec», that contains a set of UML packages that model the *invariant*, *dynamic*, and *static schemata* of the ODP information specification in UML (see Figure B.24). The following sub-clauses define these UML packages and their contents.

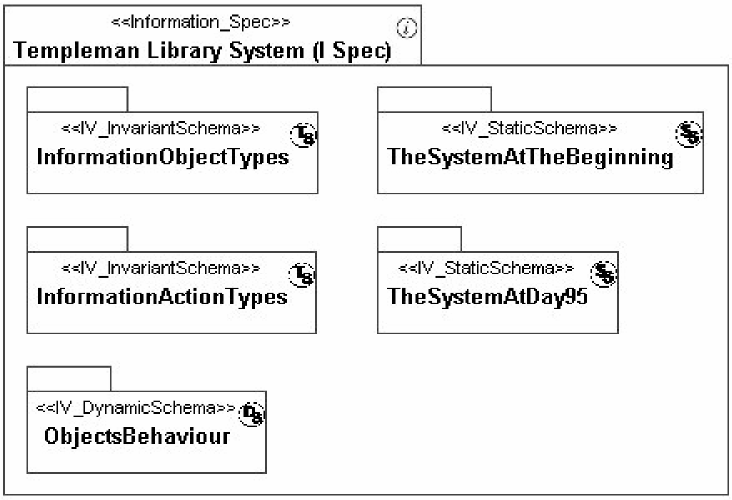



**Figure B.24 – Structure of the Information Viewpoint Specification of the Library system (excerpt)**

### B.3.2 Basic elements

From the textual regulations of the Library we can identify several main *information object types*, namely **Borrowers**, library **Items**, **Librarians**, and **Library Assistants**. In addition, a **Calendar** *object* should be in charge of representing the passage of time, and **loan** *objects* will represent the relationships between **borrowers** and **items**. Figure B.25 shows a UML class diagram with all the basic *object types* used in this information specification. UML class **PersonalParticulars** contains the personal information about the library users, librarians and assistants.

The attributes of each UML class define the information captured by this specification. Please notice that this information specification is built considering the elements of the enterprise specification described in clause B.2. The RM-ODP does not impose any methodology for the definition and use of the five viewpoints. However, for building the UML information viewpoint specifications of this particular example we have used its enterprise specifications. This approach greatly facilitates the definition of the ODP *correspondences* between the related entities that appear in the different viewpoints, and also simplifies the treatment of the *consistency* among viewpoints. Viewpoint consistency tries to detect and resolve the possibility that different viewpoints may impose contradictory requirements on the same *system*.

In particular, this information specification incorporates the information kept in the system for each user and library staff (**name**, **address**, **faculty**, etc.), and for each **Library item**: **title**, **author**, **ISBN** or **ISSN**, its physical **location**, and its current **status**: **on-loan**, **free**, **withheld** (if the circulation of the item has been temporarily withheld), **disposed** (if the item has been sold, donated, recycled, or discarded), **missing** (if the item is missing), or **other** (in case the item is in a status not contemplated by any of the previous options).

Some general and common parameters about the library are represented by another *information object* (**Library**), with details about the daily rates to be charged to late-returners, the credit obtained by collecting the payment of the fines, whether the library is open or not to the public, and the current loan limits and periods for the different kinds of users. This is a *composite information object* that also includes information about the current Library **Users**, **Items** and outstanding **Loans** in the system. This information is represented in terms of UML composition associations between this *object* and the **Librarian**, **Assistants** and **Calendar** *objects*.

The UML classes in Figure B.25 map to the ODP *information object types* of the library system. Since these classes represent the information managed by the computerized system about such *object types*, there is no need to define a **LibrarySystem** class that represents such a computerized system.

A UML class model expresses constraints on the kinds of objects and the kinds of links that can appear in a valid object configuration. Restrictions on the classes, their attributes, and the multiplicity of the associations can also be used to impose further constraints on the system objects — in the information viewpoint, such constraints correspond to the *invariant schemata*, and will be described in Clause B.3.3.

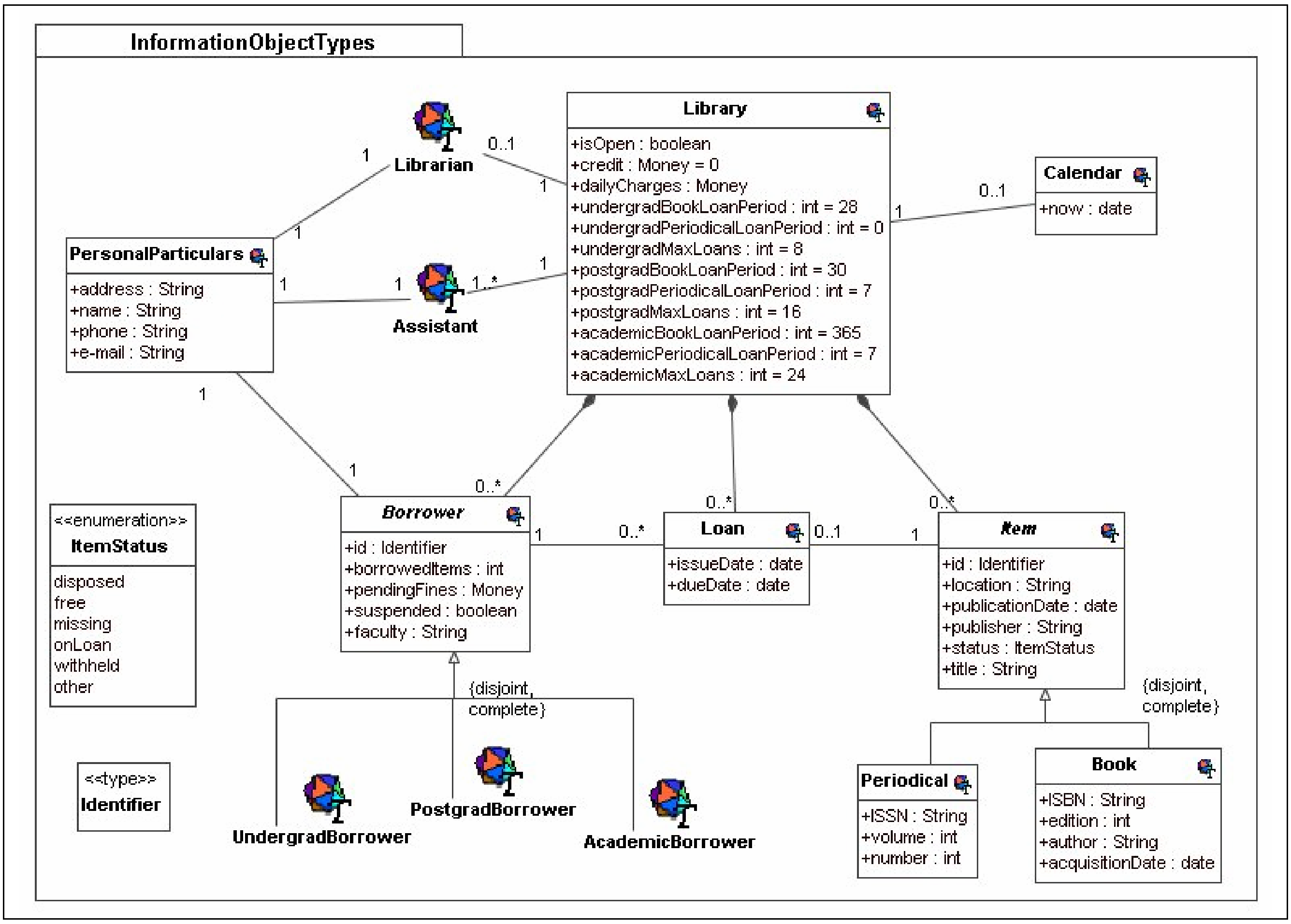



**Figure B.25 – Object types of the information viewpoint specification of the Library system**

Figure B.26 shows another UML package with the *action types* supported by the *information objects* of the system.

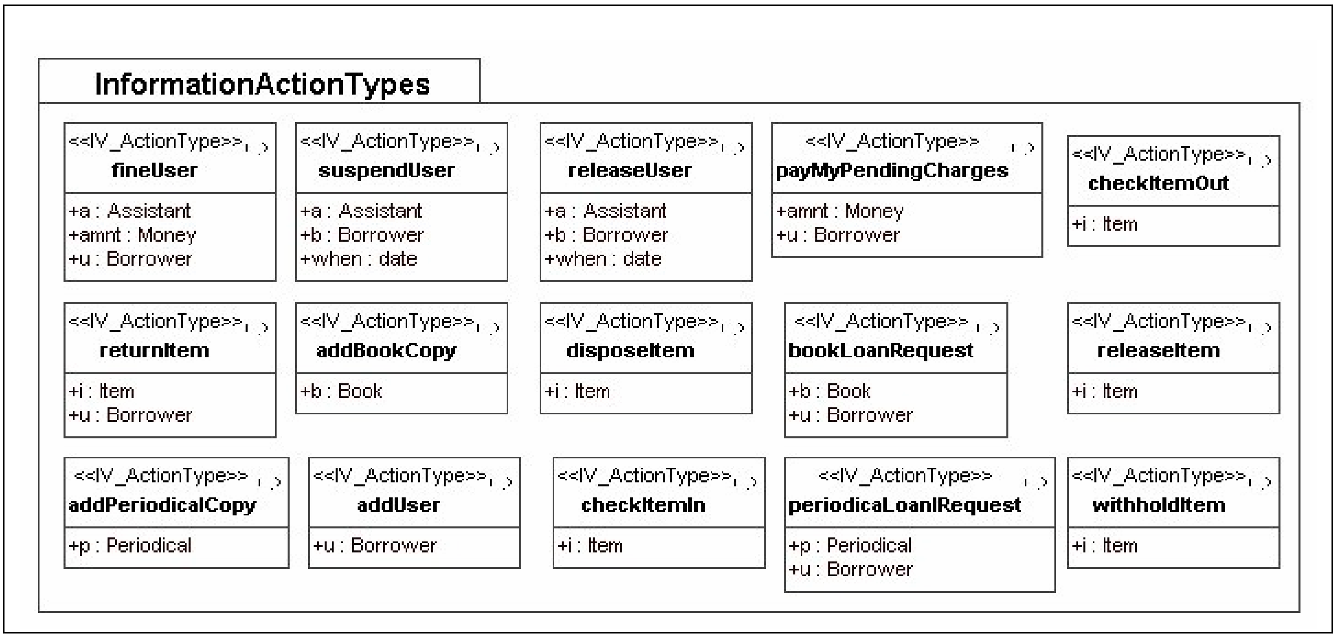



**Figure B.26 – Action types of the information viewpoint specification of the Library system**

According to the library textual regulations, and the *processes* and *actions* defined in the enterprise viewpoint of the system, the following *action types* can be identified:

- a user requests a library assistant to borrow a book (**bookLoanRequest**) or a periodical (**periodicalLoanRequest**). Such request can be accepted or denied;
- an assistant checks out a Library item before giving it on loan to a borrower (**checkItemOut**);
- a borrower returns a borrowed item (**returnItem**);
- an assistant checks in a Library item when it is returned (**checkItemIn**);
- an assistant fines a borrower (**fineUser**);
- a borrower pay charges (**payMyPendingCharges**);
- a librarian suspends a borrower (**suspendUser**);
- a librarian lifts the suspension to a borrower (**releaseUser**);
- a librarian restricts the circulation of an item (**withholdItem**);
- a librarian restores the circulation of an item (**releaseItem**);
- a librarian disposes of an item (**disposeItem**);
- a librarian adds a copy of a book or a periodical to the Library collection (**addBookCopy** or **addPeriodicalCopy**, respectively);
- a librarian adds a borrower to the system (**addUser**).

As information *action types*, they all will be represented in this example by UML signals, which will trigger the state changes in the state machines of the objects. These state machines will specify the *dynamic schemata* that will describe the *state* changes caused in the system by such *information actions*. Those *dynamic schemata* will be described later in Clause B.3.5. Attributes of the UML signals model the information conveyed by the ODP *interactions* represented by such signals.

Once we have defined the main *information object types* of the system, and the possible *actions* that may take place, the way the library system works (from the perspective of the information viewpoint) needs to be defined in terms of how information is processed. *Invariant*, *static* and *dynamic schemata* are the mechanisms defined for that purpose.

### B.3.3 Invariant Schemata

In ODP, an *invariant schema* is the specification of the *types* of one or more *information objects* that will always be satisfied by whatever *behaviour* the *objects* may exhibit. The following are examples of *invariants* that can be defined for the Library system:

1. Undergraduate students cannot borrow more than eight books.
2. Undergraduate students can borrow books for up to four weeks.
3. Undergraduate students cannot borrow periodicals.
4. Postgraduate students cannot borrow more than 16 items (books or periodicals).
5. Postgraduate students can borrow books for one month.
6. Postgraduate students can borrow periodicals for one week.
7. Academic staff cannot borrow more than 24 items (books or periodicals).
8. Academic staff can borrow books for one year.
9. Academic staff can borrow periodicals for one week.
10. Library users have unique identifiers in the system.
11. Library items should have unique identifiers in the system.
12. No item can be simultaneously referenced by two loans in the system.
13. There should be exactly one **Calendar** object and one **Librarian** in the system while the library is open.
14. The number of pending loans in the system should be consistent with the sum of the values of attribute **borrowedItems** of all the **Borrower** objects.
15. Borrowers who do not pay their fines will be eventually suspended.
16. Suspended borrowers who settle their debts will eventually be released, and thier borrowing rights restored.

Please note how some of these *invariants* have been incorporated into the UML class diagram that describes the system structure (shown in Figure B.25) in terms of the multiplicity of the association ends. This is the case, for instance, of invariant 12 (which is represented by a multiplicity “1” in the corresponding association end).

Other invariants can be naturally represented in UML by associating OCL constraints to some of the UML elements of the specification. For example, UML constraints can be added to objects to the **Library** class to represent the maximum number of loans and periods of loans permitted for every kind of borrower (hence representing invariants 1 to 9):

```
-- Invariants 1 to 9
context Library inv:
(undergradMaxLoans = 8) and (undergradBookLoanPeriod = 28) and (UndergradPeriodicalLoanPeriod = 0) and
(postgradMaxLoans = 16) and (postgradBookLoanPeriod = 30) and (postgradPeriodicalLoanPeriod = 7) and
(academicMaxLoans = 24) and (academicBookLoanPeriod = 365) and (academicPeriodicalLoanPeriod = 7)
```

Invariants 10 and 11 impose that the identifiers of users and **Library** items should be unique in the system. Both invariants can be expressed in terms of OCL constraints on the **Library** class:

```
-- Invariant 10
   context Library
   inv: self.items->forAll( itm1,itm2 | itm1.id <> itm2.id)
-- Invariant 11
   context Library
   inv: self.users->forAll( usr1,usr2 | usr1.id <> usr2.id)
```

Invariant 12 imposes that no item can be simultaneously referenced by two loans in the system. As mentioned before, this invariant has been implemented by a multiplicity “1” in the corresponding association end.

Invariant 13 states that there should be exactly one Calendar object and one Librarian in the system while the library is open.

```
-- Invariant 13
   context Library
   inv: isOpen implies (self.Librarian->size() = 1) and (self.Calendar->size() = 1)
```

Invariant 14, which imposes a consistency check on the system, can be also expressed as an OCL constraint on the **Library** class:

```
-- The number of pending loans should be consistent with the sum of the number of pending loans of each user.
   context Library
   inv: self.users.borrowedItems->sum() = self.loans->size()
```

Other invariants may need to be expressed differently. In fact, invariants 15 and 16 can be considered as predicates in a given discrete linear temporal logic that imposes some fairness constraints. OCL is not expressive enough to specify them, although we can always either use a textual description of such predicates, or use any other notation (in this case we will consider an extension of OCL with the temporal logic operators “**always**” and “**eventually**”):

```
-- Invariant 17: Borrowers who do not pay their fines will eventually be suspended.
   context Borrower
   inv: eventually always (fines = 0) or always eventually (suspended = true)
-- Invariant 18: Suspended borrowers who have paid their fines will eventually be released
   context Borrower
   inv: eventually always (fines > 0) or always eventually (suspended = false)
```

Finally, other OCL constraints may represent invariants that express well-formedness rules of the model. For instance, the following constraint restricts the valid values of Loan objects:

```
context Loan
   inv: issueDate <= dueDate
```

Similarly, other OCL expressions can help determining the value of some of the system attributes, e.g., when the library is open:

```
context Library
inv: (hour(self.Calendar.now) >= 8) and (hour(self.Calendar.now) <5) implies self.isOpen = true
```

### B.3.4 Static Schemata

*Static schemata* provide instantaneous views of information, for example at system initialisation, or at any other specific moment in time that is relevant to any of the system stakeholders. This specification of the instantaneous state of the *objects* is precisely the one provided by UML object diagrams (also known as *snapshots* in some UML dialects).

For instance, the UML object diagram shown in Figure B.28 represents a *static schema* that models the state of the system at a moment in time (namely, day 95 after its creation), in which there are only two **Borrowers** (John and Mary), one **Librarian** (Emerald), two **Assistants** (Eve and Pete), three **Books** (one copy of Ulysses and two copies of

Dubliners), and one **Periodical** (yesterday's edition of The Times). There is only one **Loan** (Mary borrowed Ulysses one month ago).

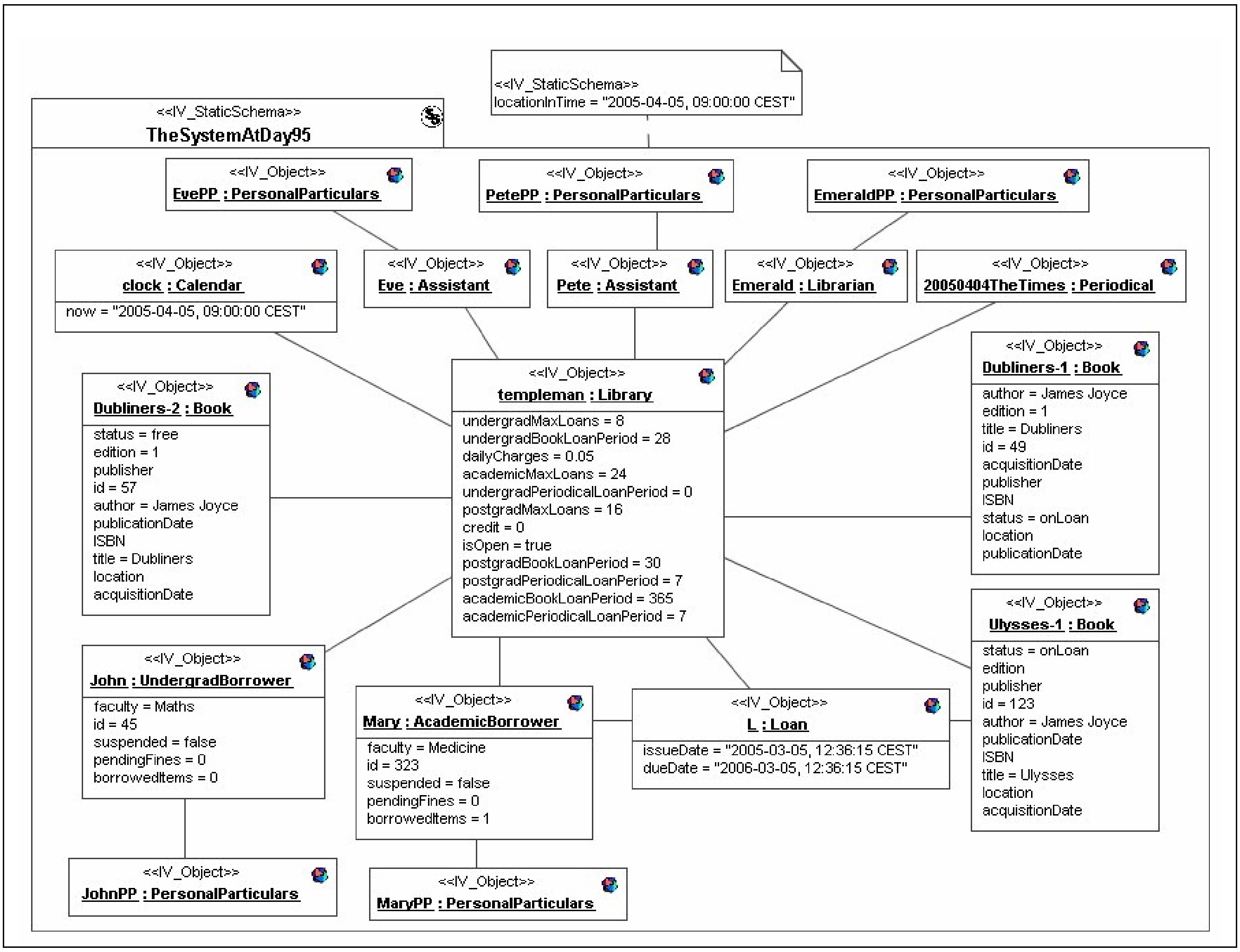



**Figure B.28 – Static schema with the configuration of the Library system at day 95**

Similarly, the following UML package represents the initial state of the system, when there are no **items**, **borrowers**, and **loans**. There are, however, one **clock**, one **assistant**, and one **librarian** at that moment in time. At least one **Assistant** should be present in order for such a configuration of *objects* to respect the *invariant schemata* specified by the multiplicity of the UML associations in the UML class diagram shown in Figure B.25.

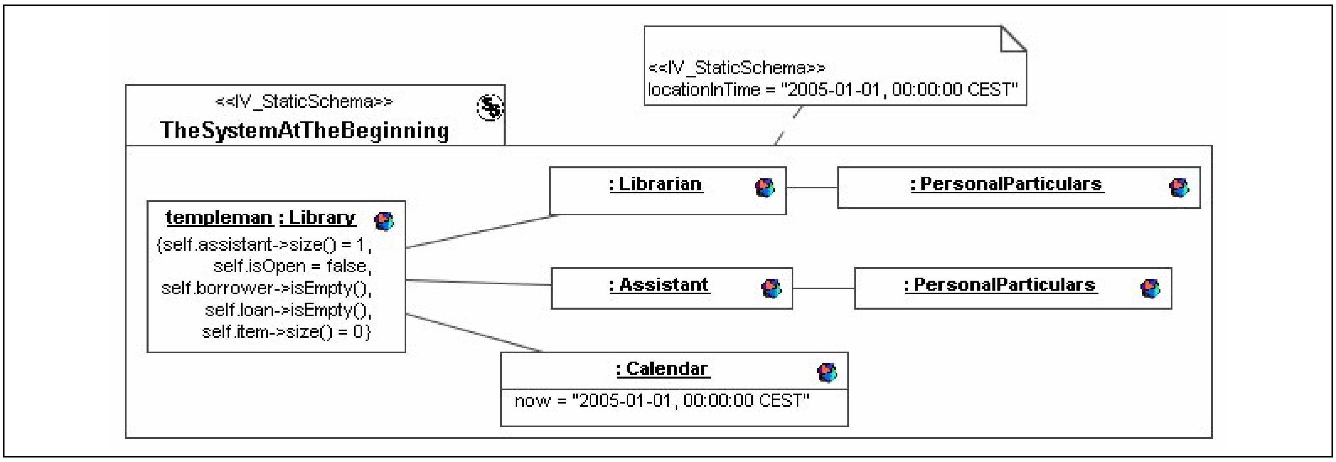



**Figure B.29 – Static schema with the initial state of the Library system**

Please note as well how the constraints on the **Library** object explicitly specify the cardinality of the links.

### B.3.5 Dynamic Schemata: Description of the system behaviour

The way the system evolves is dictated by the *behaviour* of the *objects* of the system, which in the information viewpoint is expressed in terms of a set of *dynamic schemata*. They describe the allowed *state changes* of the system or of any subset of its constituent *information objects*.

This subclause presents *dynamic information schemata* that describe changes of state associated with the *action types* identified in B.3.2. In this case, such *action types* have been represented by UML signals.

NOTE – It is worth noting here that some authors have proposed the use of UML operations for representing action types. However, this approach presents some limitations. For example, it forces actions to be owned by one object (i.e., the object to which the operation is assigned to). In general, it may be the case that more than one ODP object might be related to a single action, because ODP interactions are pieces of shared behaviour, with no necessarily single owner or initiator. However, the interaction model of the UML is based on message exchange between objects, which forces all UML operations to be assigned to only one object. Thus, if ODP information actions are represented by UML operations, the system designer has to decide, for every action, the object to which an UML operation representing the ODP information action type is assigned. This is in general a difficult decision, and therefore more practical applications are required in order to identify a set of guidelines or patterns to support the practising modeller assign ODP action types to UML object types.

A *dynamic schema* can be expressed in UML as a UML state machine or a UML package of the state machines of several UML objects (those modelling the *information objects* specified in the *dynamic schema*). In this case, the *dynamic schema* of the library is composed of the state machines of the objects that support the operations defined in clause B.3.2, namely the **Librarian**, **Assistant**, **Borrower**, and **Item**. Figures B.30 and B.31 show the state machines of some of these objects, for illustration purposes.

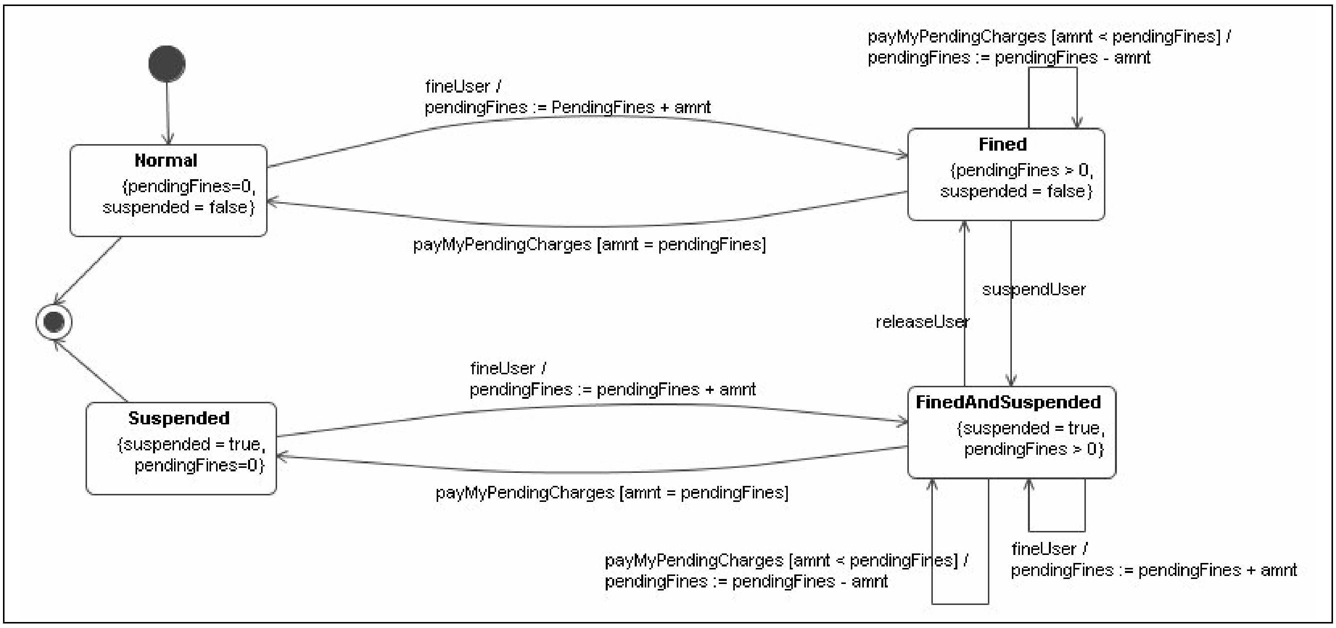



**Figure B.30 – State machine of a Borrower information object**

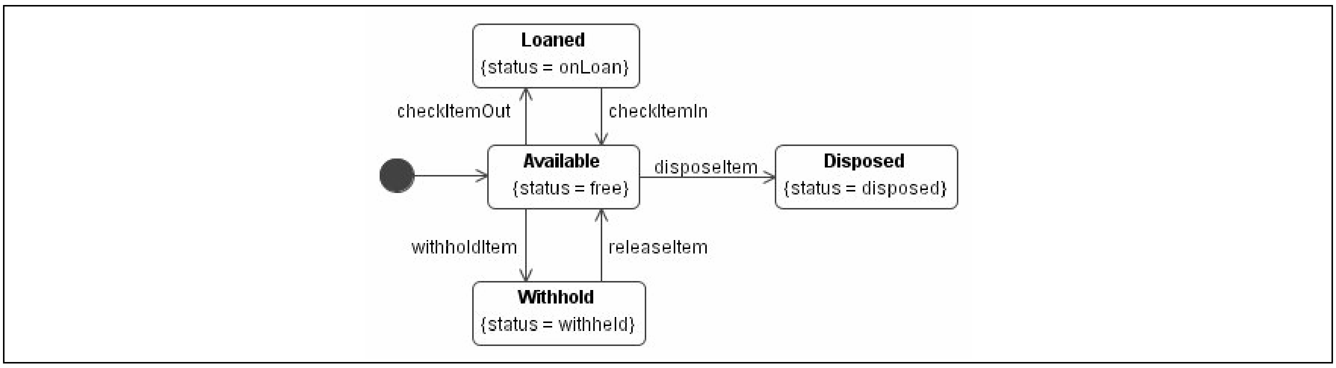



**Figure B.31 – State machine of an Item information object**

Thus, the *behaviour* of every *information objec*t is specified by a UML state machine, which describes its state changes as consequence of the occurrence of the signals that represent the possible information *actions* previously specified. These state machines model the *dynamic schemata* of the ODP information specifications. Please notice how a signal causes changes in all state machines that define a transition for it. In this way we can model, in a natural manner, the fact that an ODP *interaction* may cause a state change in all objects related to that *interaction*, i.e., an ODP *interaction* is a piece of shared behaviour. This would be very difficult to express if ODP *interactions* were mapped to UML operations on objects.

Note as well that the previous diagrams show not only the effect of the *actions* on the corresponding *information objects*, but also the *states* in which the *actions* are allowed, serving as pre- and post-conditions for those *actions*.

### B.3.6 Correspondences between the Enterprise and the Information specifications

Temporary Note – NBs are requested to provide text for this clause in the example, indicating, for instance, how the enterprise polices of the Library can be related to the different schemata of the information viewpoint; or how computational messages and operations can be mapped to and from information actions. Moreover, correspondences should be defined, for the information viewpoint, at least between its concepts and the concepts of the enterprise and computational viewpoints, and vice-versa.

## B.4 Computational specification in UML

### B.4.1 Overview

The computational viewpoint is concerned with functional decomposition of an ODP system in distribution transparent terms. A computational specification defines units of functions as computational objects, and the interactions among those computational objects, without considering their distribution over networks and nodes.

This clause concentrates on the computational specification in UML of the borrowing process of the Library system.

### B.4.2 Computational objects and interfaces

To represent the computational specification for the Templeman Library, we need to identify first the computational elements that participate in the borrowing process. These elements (i.e., computational objects and interfaces) are instantiated from their corresponding computational templates. In UML, we represent the system structure using a component diagram that describes the computational object templates and the computational interface templates at which these objects interact.

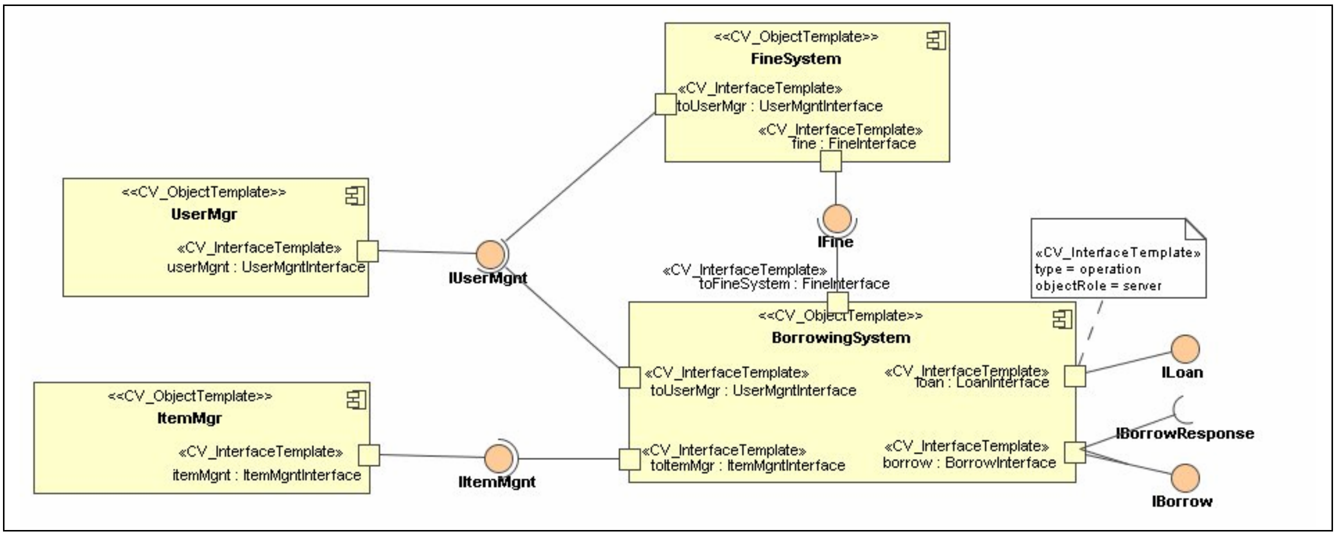



**Figure B.32 – Component diagram with computational object templates and interface signatures of the system**

Figure B.32 shows the structural diagram for the Templeman example. This model does not try to be exhaustive, just to depict how the main computational elements may be represented in UML 2.0 with the ODP Profile.

We consider four different computational objects directly related to the library elements that need to be managed:

(a) a manager (**UserMgr**) for each user;

(b) the system that controls the fines applied to users which exceed the borrowing period (**FineSystem**);

(c) the objects that manage the library items (**ItemMgr**); and

(d) the computational object that coordinates the whole borrowing process (**BorrowingSystem**).

These objects interact which each other and with their environments at computational interfaces, which are instantiated from their corresponding interface templates. In this case, five different interface templates have been defined; all of them correspond to operational interfaces. Interface templates are represented in Figure B.32 as UML ports. For readability reasons, balls and sockets are used to represent interfaces signatures.

In this example, only operation computational interfaces have been defined. Figure B.33 shows a detailed description of the interactions defined at interface **UserMgntInterface**. ODP operations are mapped to UML operations. This approach is very useful when the exchange of information between objects can all be modelled in terms of operation interactions between computational objects. In this case, modelling these interactions as UML operations might be probably simpler than breaking them into their corresponding signals.

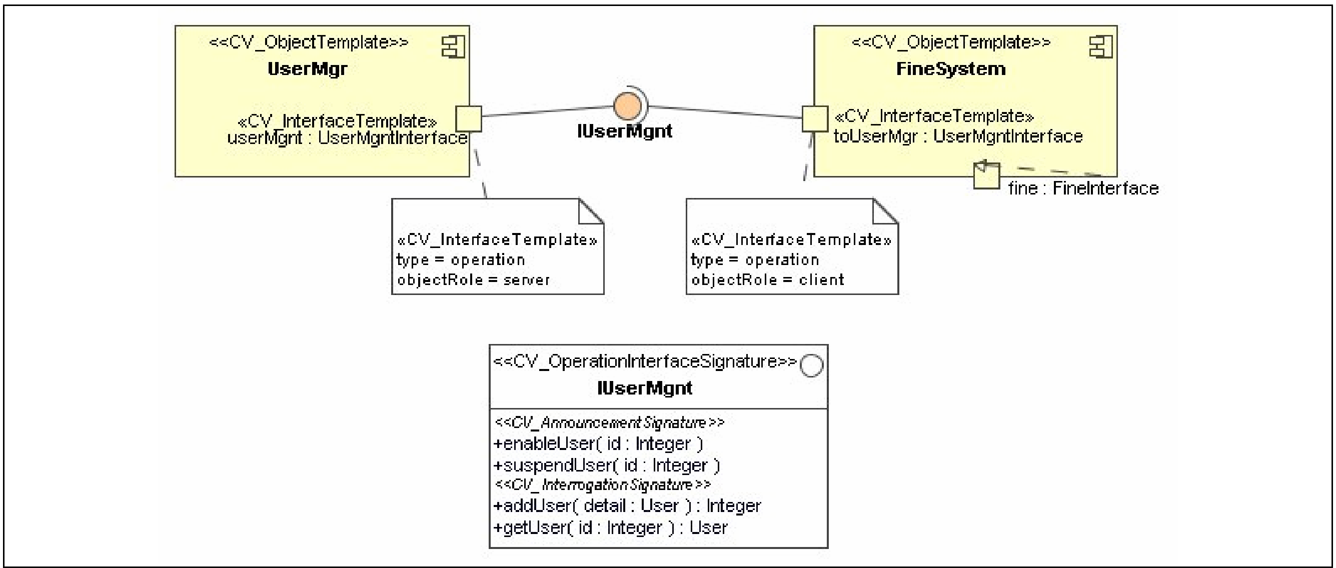



**Figure B.33 – Example of interaction signatures modelled as UML operations**

As shown in Figure B.33, interaction signatures are grouped according to their corresponding stereotypes (CV_AnnouncementSignature, CV_InterrogationSignature) inside the UML interface classifiers.

Figure B.34 shows the rest of the interface signatures. The interactions of interface signature **IBorrow** have been defined using interrogations and terminations for illustration purposes, instead of using UML operations as we do for the rest of the interface signatures.

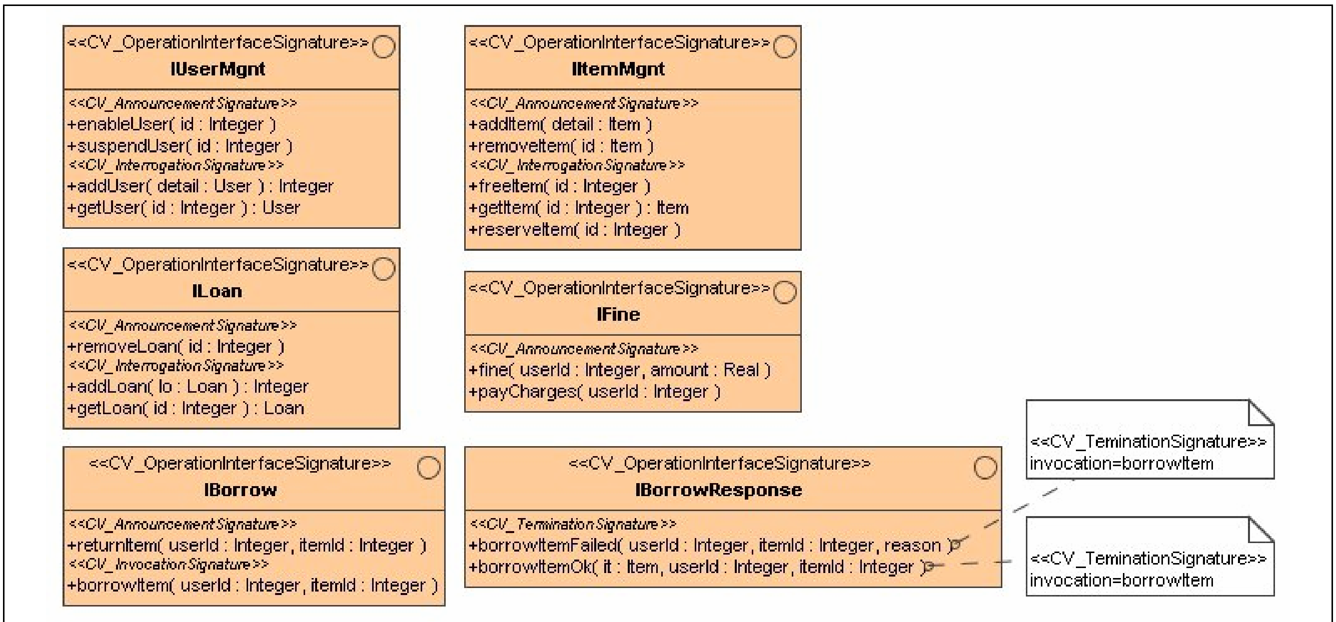



**Figure B.34nteraction signatures for Library interfaces**

### B.4.3 Behaviour

Apart from the structural aspects, we need to specify the behaviour of the computational elements of a specification. UML activity, communication, interaction and sequence diagrams might be useful to represent both the internal actions of the computational objects, and the interactions that occur between them.

In case we want to specify how objects interactions are performed, activities can be useful because they are abstractions of the many ways in which messages are exchanged between objects. This makes activities useful when the primary concern is the dependency between tasks, rather than the interaction protocols.

The activity diagram for the borrowing process is shown in Figure B.35

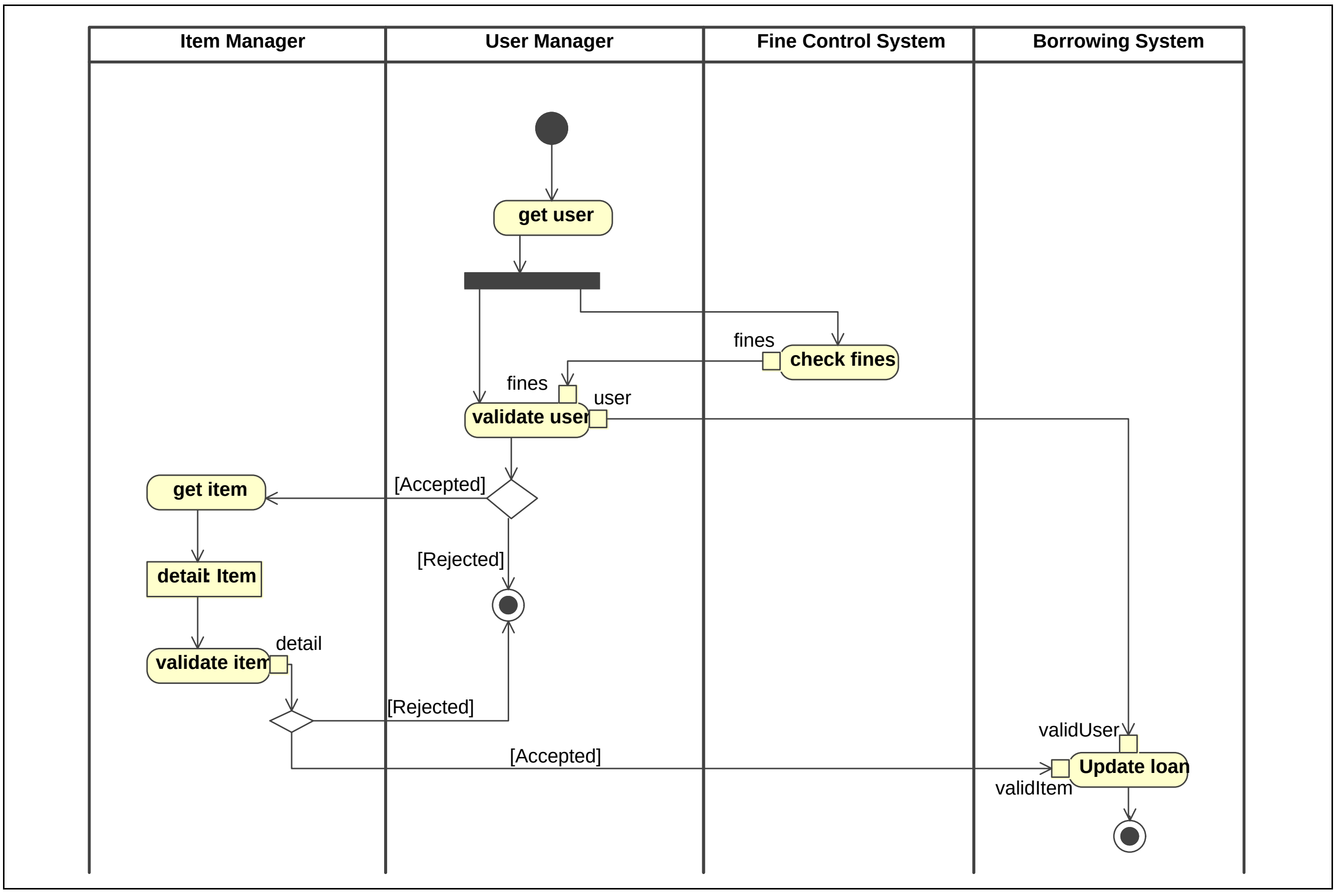



**Figure B.35Activity diagram for the borrowing process**

Alternatively, UML interaction diagrams are more appropriate when messages and interaction protocols are the focus of design, as shown in Figure B.36

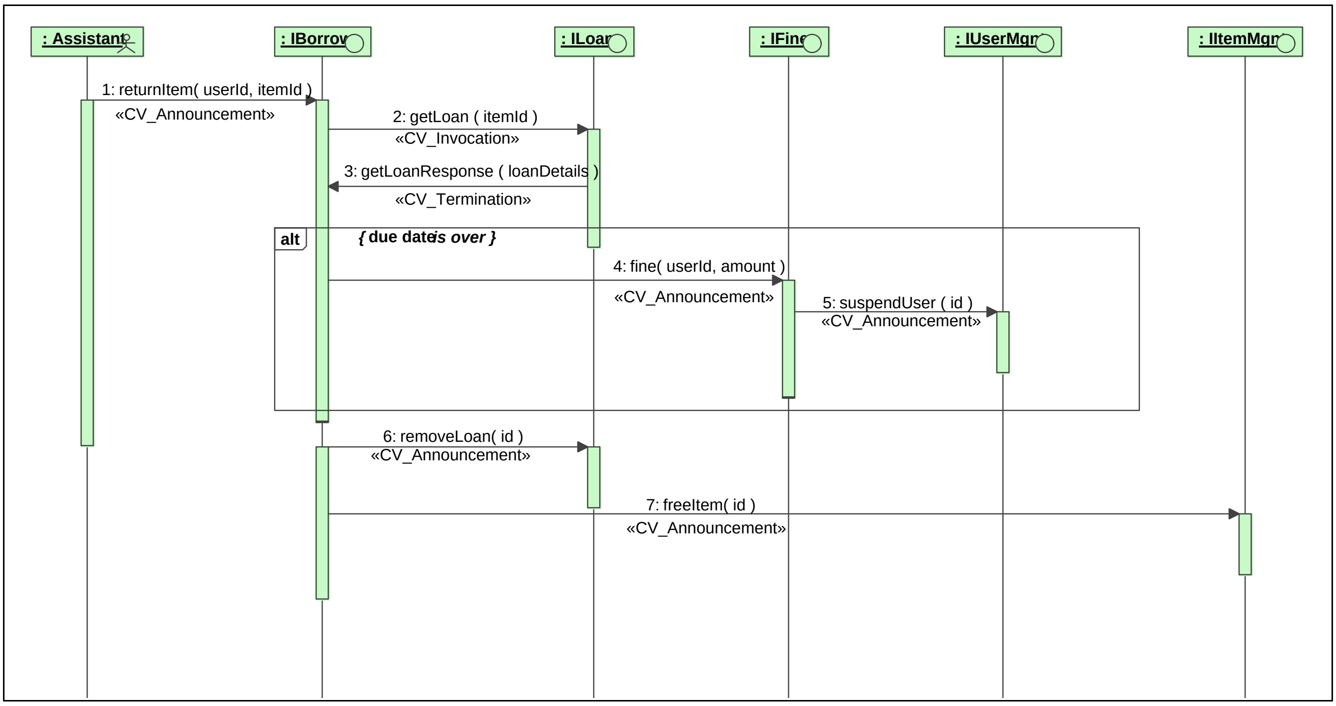



**Figure C.36 Interaction diagram for the borrowing process**

### B.4.2 Environment contracts

*Environment contracts* place constraints on the *behaviour* of *computational objects*, and usually include QoS, usage and management aspects. The ODP Reference Model does not prescribe how an *environment contract* must be specified; it just defines this concept and its basic contents.

Each system modeller might like to specify their own constraints in the way that best suits their particular application. Therefore the UML elements (and their semantics) required to model different *environment contracts* can change from one application to another. Thus, instead of incorporating this kind of concepts into the ODP-CV Profile, QoS and other extra-functional aspects of *environment contracts* may be represented by separate specialized profiles.

The possibility offered by UML 2.0 to apply multiple profiles to a package—as long as they do not have conflicting constraints—will allow system specifiers to use the QoS profile(s) of their preference, on top of the ODP Computation Viewpoint profile.

Figure B.37 shows an example in which the QoS constraints are expressed using the OMG “UML Profile for QoS and Fault Tolerance Characteristics and Mechanisms”. Notice that the application of a profile allows the use of its stereotypes, but does not necessarily require their use.

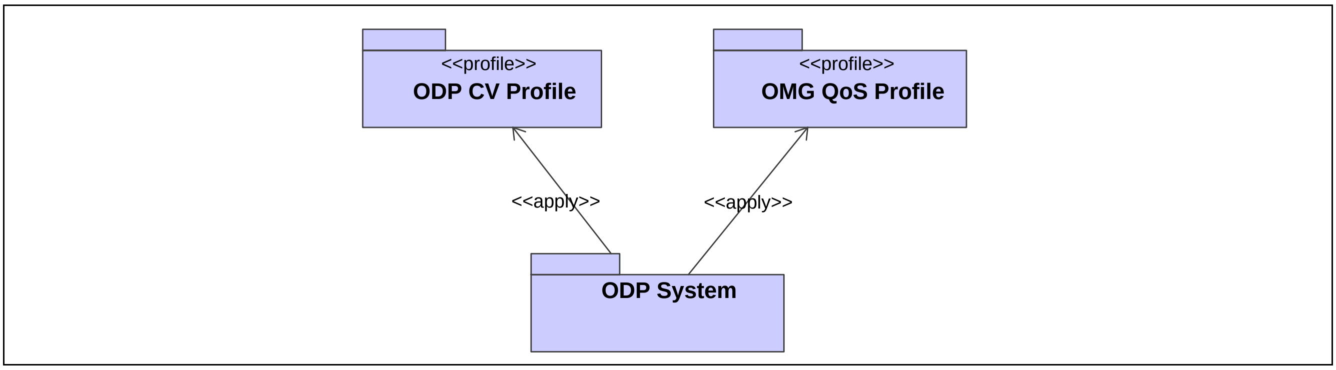



**Figure B.37 – Applying multiple profiles to an ODP system specification**

## B.5 Engineering specification in UML

### B.5.1 Overview

First we need to consider logical model for the Library example. Possible models include distributed component model and messaging system model. In this example, we will take distributed component model as logical model.

Secondly, we need to consider physical deployment model for the Library example. There are several typical architectural styles to apply to define physical deployment model, which are client-server, n-tier, model-view-controller (MVC), and service-oriented-architecture (SOA). In this example, we will take n-tier (n=4) and MVC architectural style as physical deployment model. Note that, even with this choice of architectural styles, there will be various types of node configurations, depending on requirements, such as performance, reliability, availability etc.

### B.5.2 Computational Objects

A set of computational objects, which this engineering specification will support, needs to be clarified. In this example, those are computational objects defined in B.4, which are UserManager, ItemManager, FineSystem, and BorrowingSystem. Those computational objects will be supported by corresponding basic engineering objects, which are deployed within clusters on nodes, and by engineering infrastructure, such as node, nucleus, capsule, capsule manager, cluster, cluster manager, and channel etc., supporting distribution transparencies.

### B.5.3 Node configuration

We will consider the following node configuration (Figure B.38). The basic model consists of 4-tier nodes, called ClientTier, InteractionTier, EnterpriseTier, and EISTier (EIS: Enterprise Information System) respectively. An assistant (a user of the system) will use a desktop or notebook PC, which serves as ClientTier. A request from ClientTier is sent to a server node, which serves as InteractionTier. A functional request is passed to other server node, which serves as EnterpriseTier. Finally, data persistence is taken care of by yet other server node, which serves as IntegrationTier. In this example, the following diagram shows an overview of node configuration.

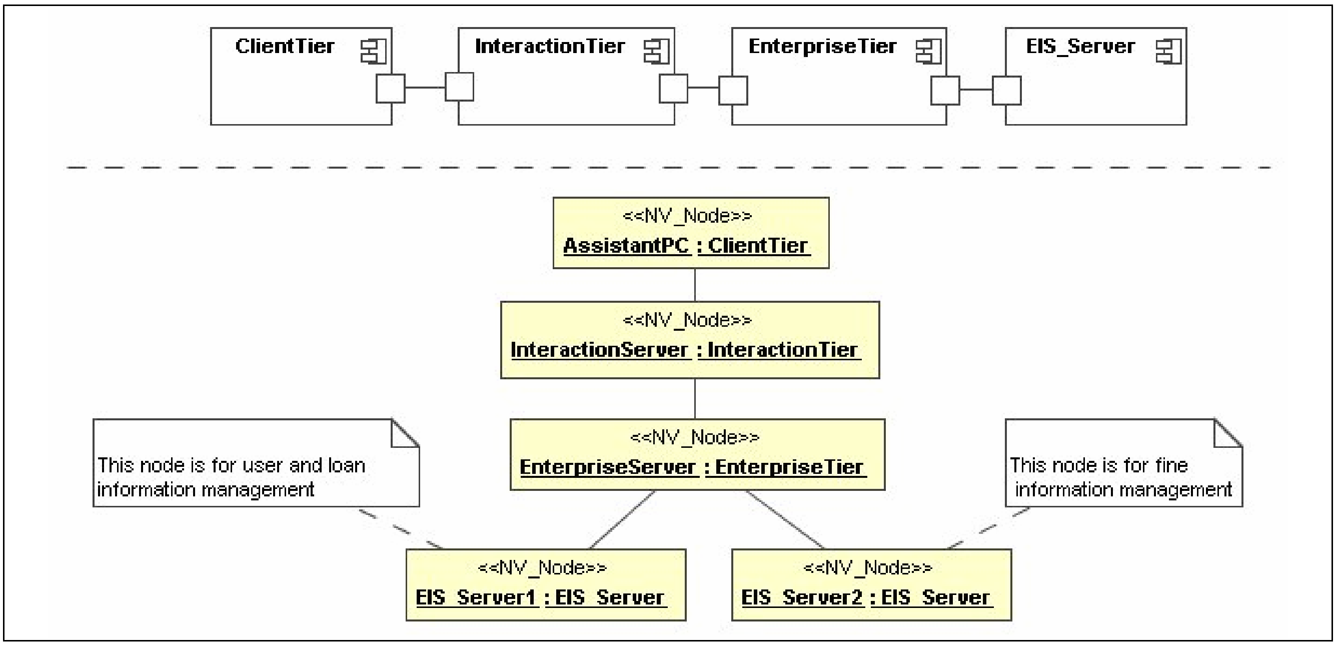



**Figure B.38 – Node configuration overview**

### B.5.4 Node structures

Each node may consists of the node itself, nucleus, capsule, capsule manager, cluster, cluster manager, basic engineering objects (BEOs), stub, binder, protocol object, and interceptor. In above node configuration, BEOs are hosted as follows:

- AssistantPC hosts BEOs for graphical user interface to access the system;
- InteractionServer does hosts BEOs necessary for n-tier and MVC architectural style;
- EnterpriseServer hosts BEOs for all four computational objects in different clusters with remaining BEOs for n-tier and MVC architectural style; and
- EIS_Server1 hosts information and information access for user and item systems, and EIS_Server2 hosts information and information access for fine system.

As an example of node structure, the following diagram shows internals of EnterpriseServer.

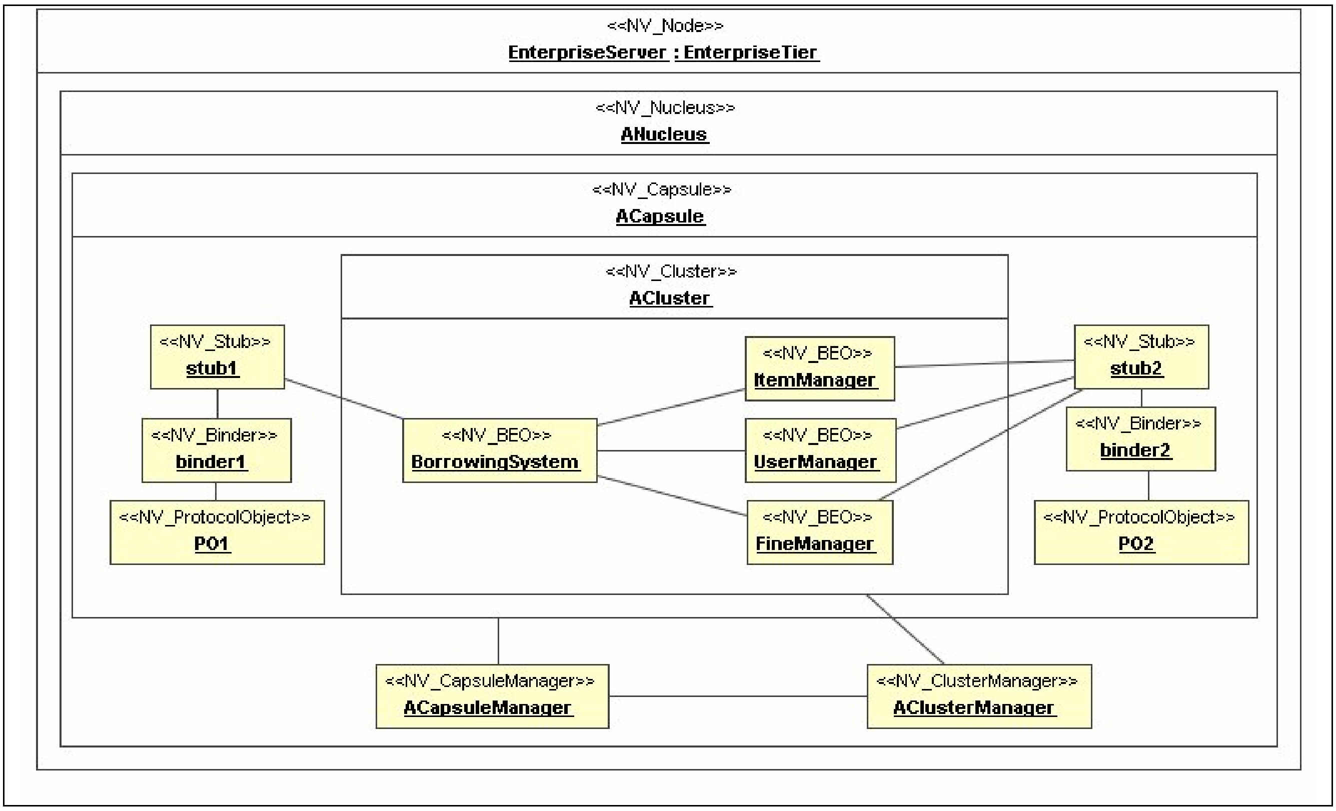



**Figure B.39 – Example: EnterpriseServer internals**

## B.5.5 Channels

In this example, four channels exist: one between AssistantPC and InteractionServer, one between InteractionServer and EnterpriseServer, and two between Enterprise Server and IntegrationServers. First channel comprises of a stub, a binder, and a protocol object of AssistantPC, and a stub, a binder, and a protocol object of InteractionServer. Since the structural aspect of a channel is defined in node model, a channel package is defined to import relevant engineering objects to construct a channel.

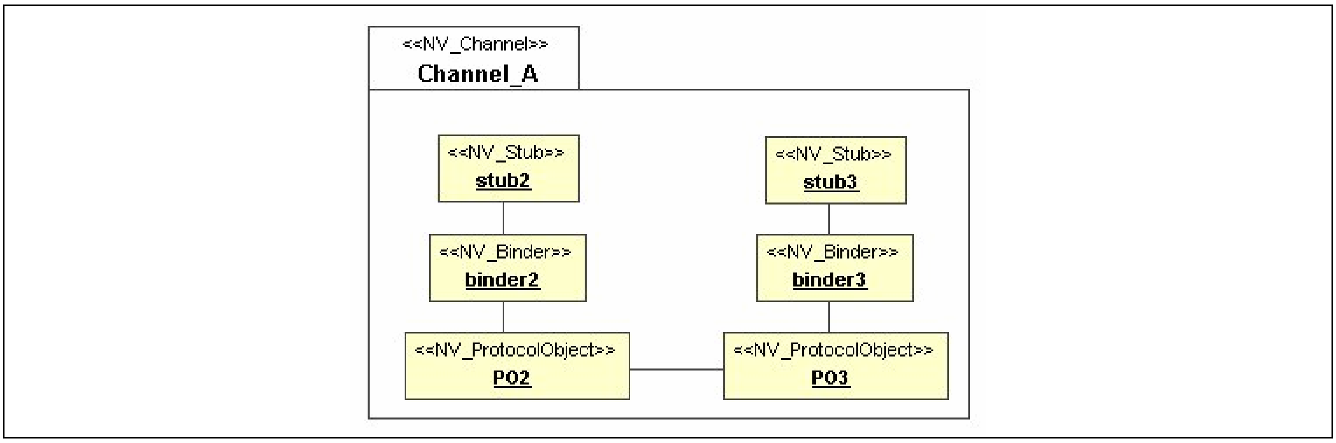



**Figure B.40 – Internals of a channel**

## B.5.6 BEO configuration

In B.5.4, several example BEOs are defined within a cluster in EnterpriseServer.

In order to define interactions between BEOs within a cluster, computational profile regarding interactions, interfaces, and signatures may be reused here. To avoid confusion, it is suggested to use stereotype for engineering language first, followed by stereotype for computational language.

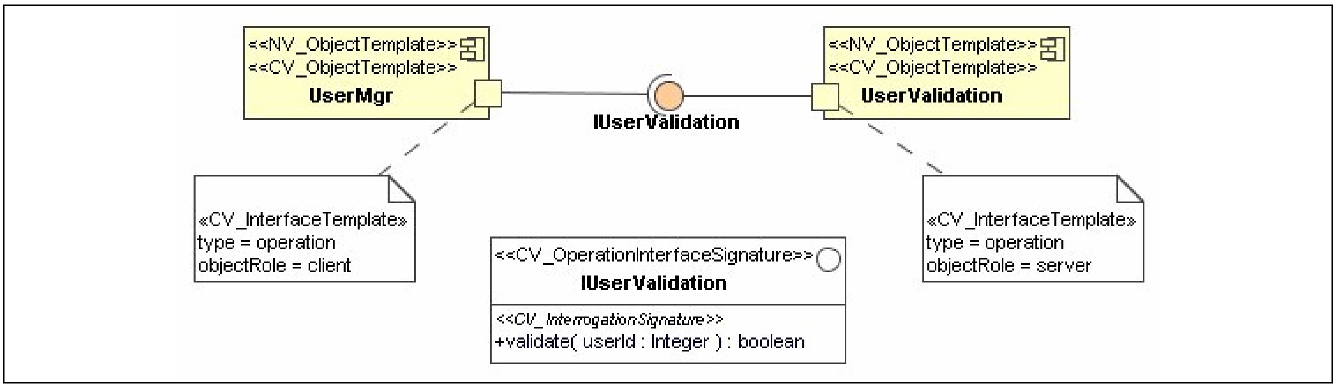



**Figure B.41 – Interactions between BEOs within a cluster**

### B.5.7 Communication Domain

As an example of domains, here is a sample of communication domain definition in UML. It is a package that accesses protocol objects which belong to the same communication domain. The domain may also have policy for controlling the communication.

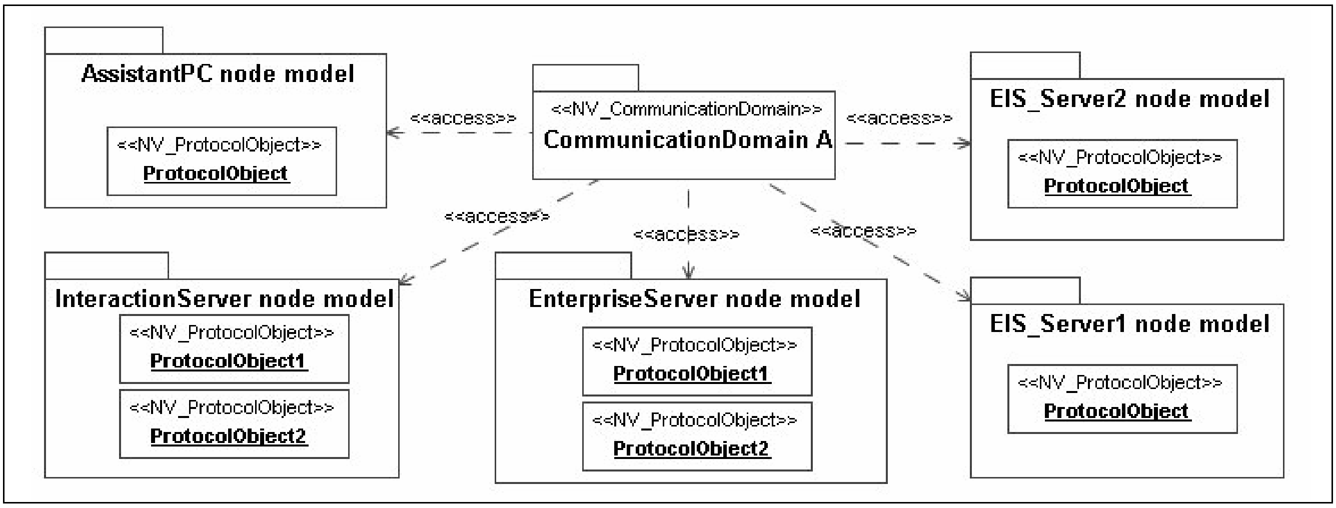



**Figure B.43 – An example of a communication domain**

## B.6 Technology specification in UML

### B.6.1 Overview

In the library example in this viewpoint, a configuration of computer systems including hardware, software, and networks connecting those systems will be described. In that configuration of technology objects, implemented standards and its implementation will be shown. If it is necessary, the process of implementation is also shown. Finally, extra information for testing, e.g. for conformance testing, will be provided.

The primary diagram for this specification is UML Deployment Diagram.

### B.6.2 Node configuration

In UML Deployment Diagram, each kind of computer node is represented with UML Node and lines are introduced to represent communication links between the nodes. Different types of network can also depicted as UML Nodes.

Figure B.44 shows the node configuration of the Library system, in two parts. The upper part of the figure describes the deployment architecture of the system by showing the different technology objects types that will be used, and how they can be connected among themselves. The diagram shows that there will be three different kinds of computing resources

(PCs, application servers and enterprise servers) and two different kinds of communication media (LAN and WAN). PCs and application servers can be connected to LANs and WANs, whilst enterprise servers can only be connected to WANs.

The lower part of Figure B.44 describes the actual system, with concrete InstanceSpecifications of the previous Nodes showing the technology objects that will comprise the system, and how they are connected.

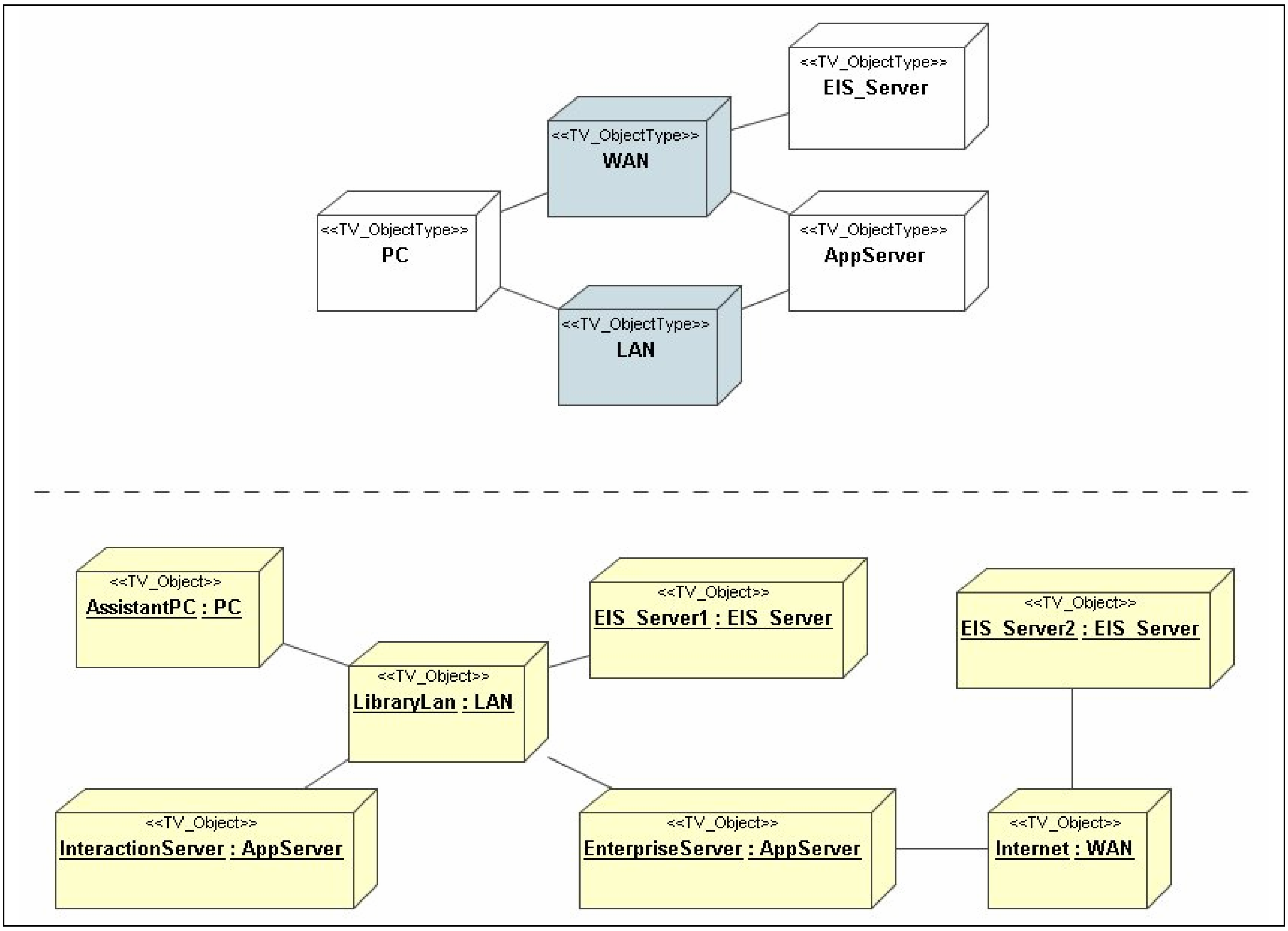



**Figure B.44 – Node configuration overview**

### B.6.3 Node structure

Technology objects, such as implementation of engineering objects, are deployed on each node. This is shown with UML Deployment Diagram including internal structure consisting of hardware elements, software elements, and network elements. There are cases where both Technology profile and standard UML Profile (e.g. ExecutionEnvironment stereotype) need to be applied to the same element.

In UML, node structures can be specified both at the type level and at the object level. The following diagram shows the internal structure of the application servers used in the Library, and therefore it is described using object types. Other diagrams can be used to show the internal structure

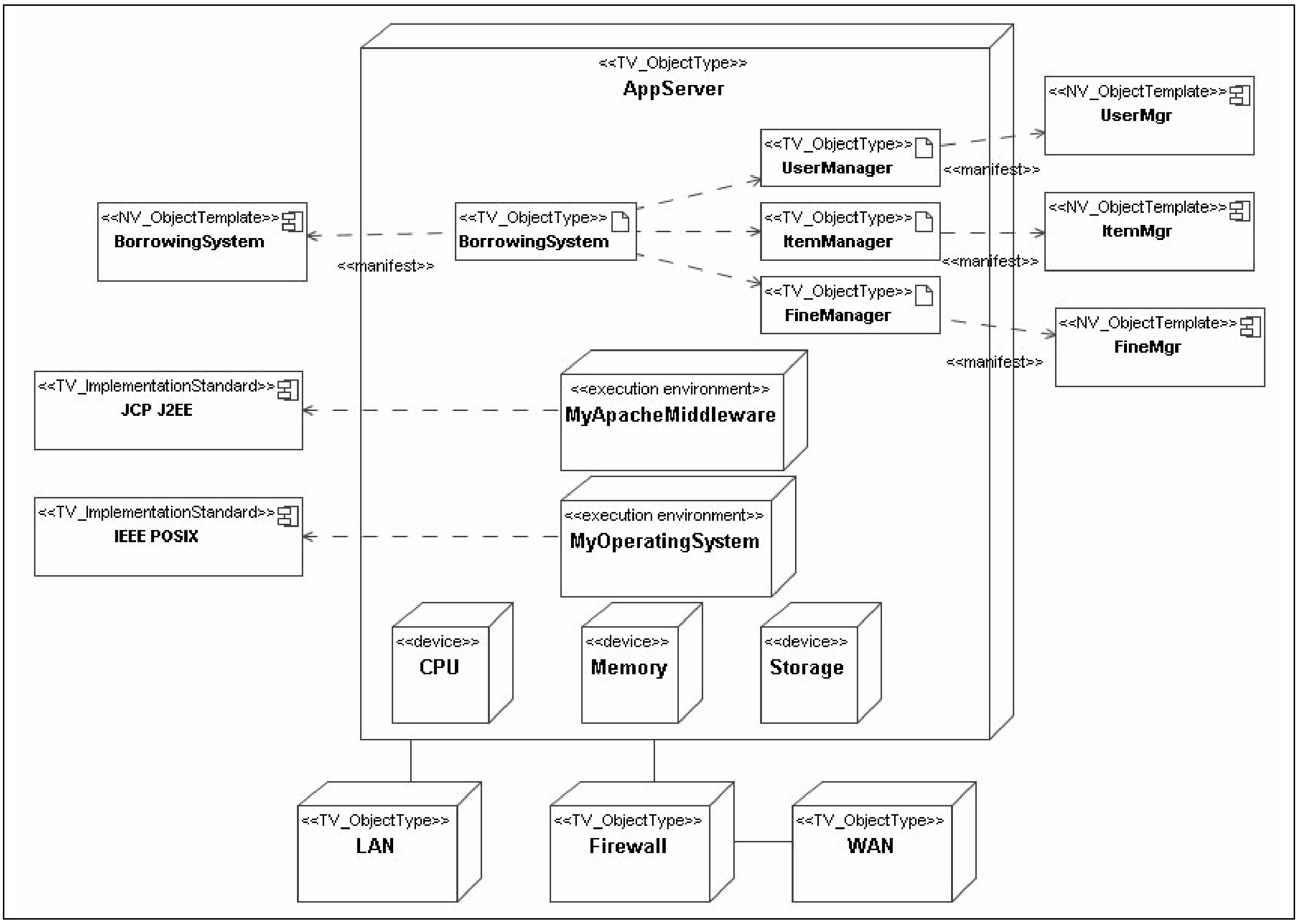



**Figure B.45 – Node structure**

### B.6.4 IXIT

The truth of a statement in an implementation can only be determined by testing and is based on a mapping from terms in the specification to observable aspects of the implementation. A test is a series of observable stimuli and events, performed at prescribed points known as reference points, and only at these points. These reference points are accessible interfaces. Four classes of reference points at which conformance tests can be applied are defined, which are programmatic reference point, perceptual reference point, interworking reference point, and interchange reference point.

IXIT is Implementation eXtra Information for Testing, which means additional information when performing a test against an implementation claiming to implement defined specification or standard. In this respect, IXIT can be attached to any technology objects for their interaction with user, other technology objects in the same node, and other technology objects in other nodes.

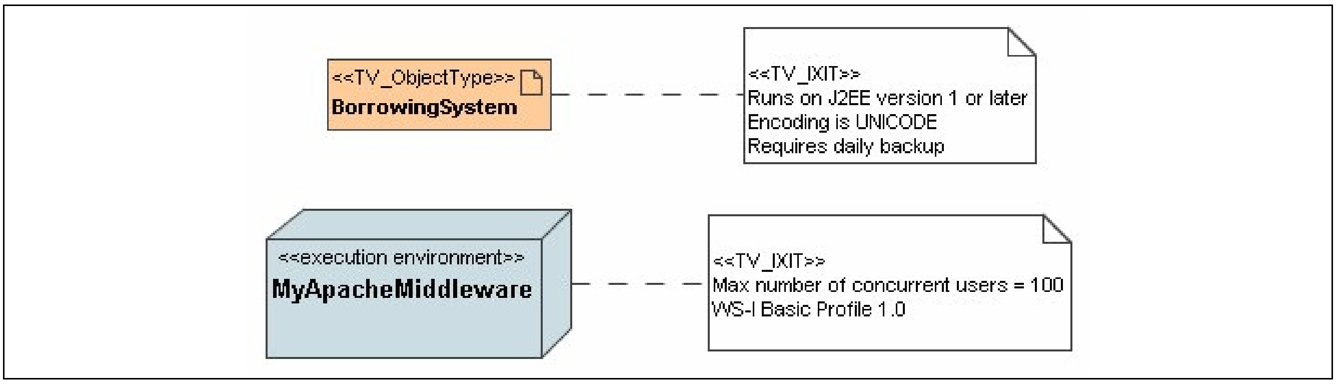



**Figure B.46 – IXIT**

# Annex C Relationship with MDA®

(This annex forms an integral part of this Recommendation | International Standard)

This annex describes the relationship of this document with the Model Driven Architecture® specified by the OMG. This annex is not normative.

## C.1 Overview of the MDA®

The Model Driven Architecture® (the MDA®) is specified in the <authoritative OMG document> (currently omg/2003-06-01, MDA Guide Version 1.0.1), and that document is the basis for the text in this subclause. The <authoritative OMG document> is the authoritative text and, in any conflict between that and this document, the former should be taken to represent OMG's position.

MDA® is an approach to system development. It is model-driven in that it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification. It provides an approach for, and enables tools to be provided for:

– specifying a system independently of the platform that supports it,

– specifying platforms,

– choosing a particular platform for the system, and

– transforming the system specification into one for a particular platform.

The three main goals of MDA® are portability, interoperability and reusability through architectural separation of concerns.

In the MDA® the term architecture of a system is a specification of the parts and connectors of the system and the rules for the interactions of the parts using the connectors, and the Model Driven Architecture® itself prescribes certain kinds of models to be used, how those models may be prepared and the relationships of the different kinds of models.

The basis for prescribing these models is the concept of viewpoint, where a viewpoint on a system is a technique for abstraction using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within that system (cf. RM-ODP Part 2). Here "abstraction" is used to mean the process of suppressing selected detail to establish a simplified model. The concepts and rules may be considered to form a viewpoint language.

The Model Driven Architecture® specifies three viewpoints on a system:

– a computation independent viewpoint that focuses on the requirements for the system; the details of the system are hidden or as yet undetermined;

– a platform independent viewpoint that focuses on the application-specific behaviour of a system while hiding the details necessary for a particular platform. A platform independent view shows the part of the complete specification that does not change from one platform to another. A platform independent view may use a general purpose modelling language, or a language specific to the area in which the system will be used;

– a platform specific viewpoint that combines the platform independent viewpoint with an additional focus on the detail of the use of a specific platform by a system.

In this context, a platform is a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented.

Corresponding to these viewpoints the MDA® prescribes three kinds of model:

– A computation independent model: a view of a system from the computation independent viewpoint. A CIM does not show details of the structure of systems. A CIM is sometimes called a domain model.

  NOTE – To specify a CIM one can use a domain language, but also a UML model that expresses the domain semantics using a domain vocabulary.

– A platform independent model: a view of a system from the platform independent viewpoint. A PIM exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms of similar type.

– A platform specific model is a view of a system from the platform specific viewpoint. A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform.

Platform independence is the quality that the model is independent of the features of a platform of any particular type. A very common technique for achieving platform independence is to express a system model in terms of a technology-neutral virtual machine. A virtual machine is defined as an interpreter of the set of rules expressed in a language or model. Given a model and a state of the environment, it determines unambiguously what actions are possible and which of them are obligatory. An implementation of a virtual machine will generally execute the model as a program (or a specialised implementation might answer questions about the model). A virtual machine is effectively an idealized platform, and any model is expressed by the way it will be interpreted by the virtual machine. But such a model is platform independent with respect to the class of different platforms on which the virtual machine has been implemented. This is because such models are unaffected by the details of the underlying platform and, hence, fully conform to the criterion of platform independence.

The MDA® also introduces a platform model providing a set of technical concepts that can express the different kinds of parts that make up a platform and the services provided by that platform. It also provides, for use in a platform specific model, concepts representing the various ways the platform can be used by an application. The aim of the platform model is to provide a specific implementation of the PIM virtual machine, effectively constraining the PIM to PSM transformation to target the specific platform.

Model transformation is the process of converting between two models describing different aspects or levels of detail of the same thing. A model transformation may be partly automated in a modelling tool, but it will normally also involve design choices by the developer and, possibly, some manual activity.

A tool builder may be primarily concerned with either the maintenance of a viewpoint specification in isolation or the maintenance of a correspondence (or transformations) between viewpoints. Thus tools may be concerned solely with a CIM, PIM or PSM, or they may be concerned with managing the correspondence between CIM and PIM, or between PIM and PSM. They may be concerned with a specific instance of the correspondence, or with the rules applicable in a broad class of use cases – that is, with the transformation as a reusable item. A particular tool may play any or all of these roles.

## C.2 Relationship of this document with the MDA®

A system specification that is compliant with this document may satisfy the requirements of the MDA®. Specifically:

- The enterprise specification is related to a computation independent model (CIM); a CIM may be provided by an enterprise specification, together with relevant parts of an information specification.
- The information and computational specifications together form a (set of) platform independent model(s) (PIM). Clause 7 (Computational Language) of RM-ODP Part 3 specifies a virtual machine that is the basis for platform independence.
- The engineering specification constitutes a model of the system that specifies support for relevant ODP transparencies (e.g., distribution). It may or may not make use of the functionality provided by the specific vendor technology defined by the technology specification. This model is platform specific with respect to the provision those transparencies, but still platform independent with respect to the particular choices of technology to implement the system.

  NOTE – The engineering language provides a series of templates for the concepts that are the elements of the computational virtual machine. The result of applying these templates is the specific executable form of the system, and is rarely made explicit.
- The technology specification defines the deployment of hardware and software (including network elements) in the system, and does not have an equivalent in the MDA.

The correspondences between the viewpoint specifications constrain the transformations by means of which one model is derived from another.

Figure C.1 illustrates the relationships described above.

Temporary Note – The figure has the merit of reflecting the text. There was no full agreement at the Málaga meeting on the precise nature of the relationships between enterprise specifications and MDA models.

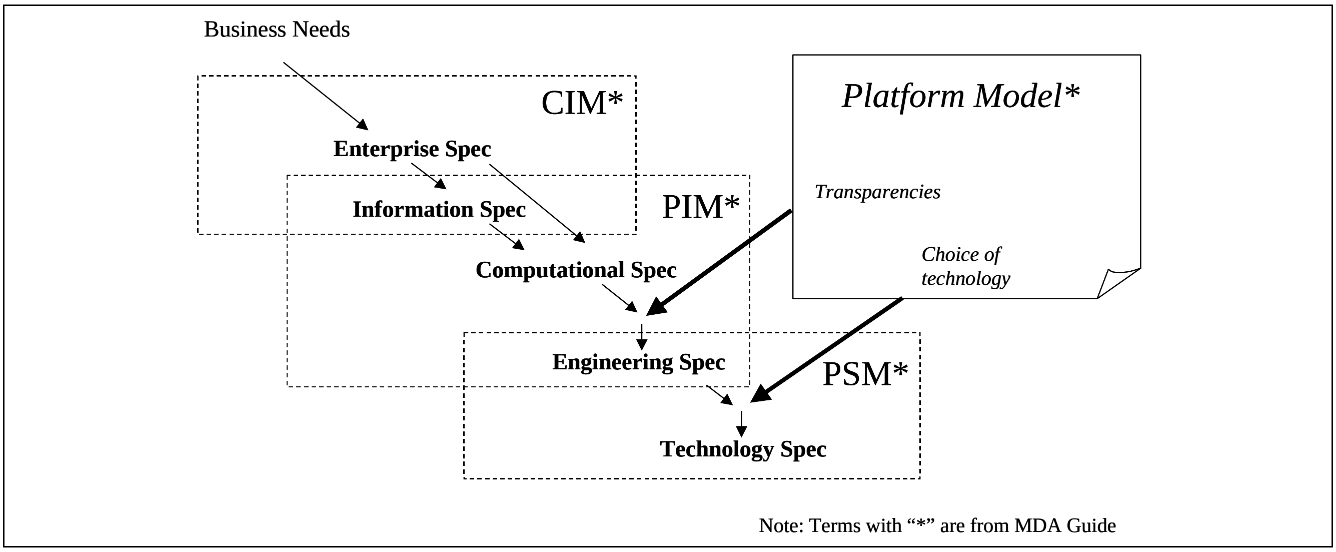



**Figure C.1 – ODP system specifications and MDA® models**

# Annex D Architectural Styles

(This annex forms an integral part of this Recommendation | International Standard)

This annex describes some common architectural styles used in the design of distributed systems, and provides the corresponding UML Profiles. This annex is not normative.

## D.1 Introduction

By observing that some problems have common answers, software architects started to codify those common architectural solutions on what they called architectural styles. As architectural styles are a way to model easily how distribution is realized, they can be leveraged to make engineering deployment model easier and more practical.

The purpose of this annex will be to describe some of the common architectural styles and provide the associated UML Profiles. The architectural styles that will be described are client-server, n-tier, Model-View-Controller (MVC), and Service-Oriented Architecture (SOA).

## D.2 Distribution Styles

### D.2.1 Client-Server

Client-server architectural style, also known as two-tier, consists on distributing the system developed between two physical nodes, the client and the server. Usually the client is always responsible of the presentation part and the server is always hosting the database. The business and data access logic can then be hosted by the client, by the server, or by a combination of the client and server.

A client that is only hosting the presentation logic is usually called a “thin client” by opposition to a “fat client” which hosts presentation, business and eventually data access logic.

The UML profile used to describe such architectural style is shown in Figure D.1.

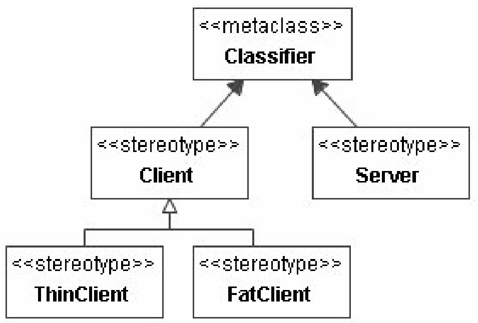



**Figure D.1 – Client-Server UML Profile**

### D.2.2 N-Tier

With the internet boom and the need to have applications always more poweful and scalable, architects introduced the notion of multi-tiered applications, enabling the possibility to run each tier on a separate physical machine.

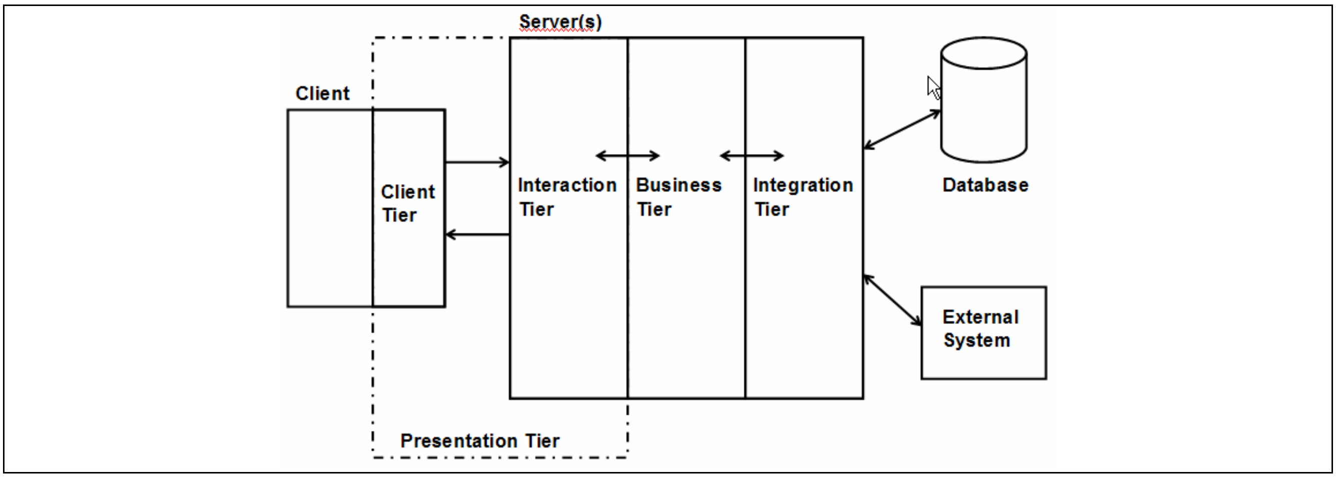



**Figure D.2– N-Tier architecture**

As shown in Figure D.2, an N-Tier system is composed of three different tiers:

– The Presentation Tier, which has the responsibility of interacting with the user. This tier contains elements that reside on both the client (Client Tier) and the server (Interaction Tier). The Client Tier is responsible for rendering the user interface and for handling user interactions. The Interaction Tier is responsible for processing client-side request and providing appropriate responses.

– The Business Tier, which has the responsability of the business logic.

– The Integration Tier, which has the responsibility of providing access to the Enterprise Information System (EIS), like databases, external or legacy systems.

Figure D.3 shows the corresponding UML Profile.

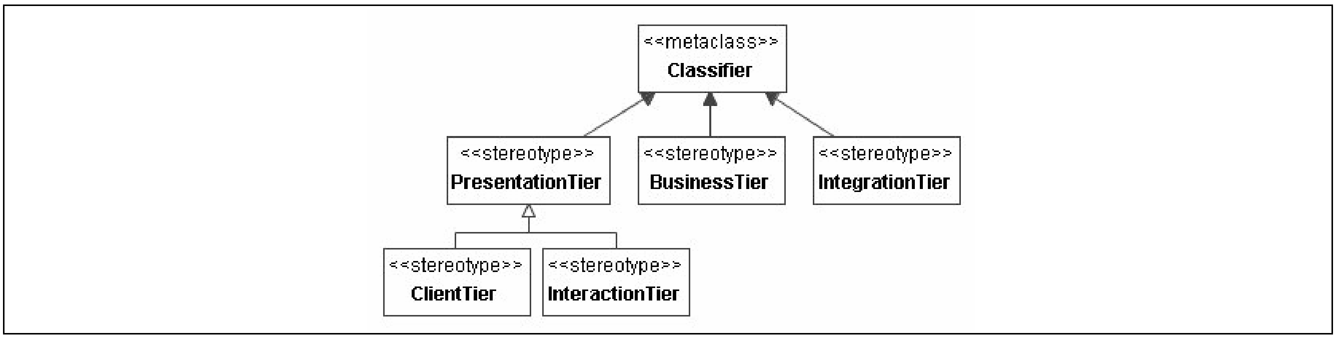



**Figure D.3– N-Tier UML Profile**

### D.2.3 Model-View-Controller (MVC)

The MVC paradigm is a way of decoupling the graphical interface of an application from the code that actually does the work. As explicitly mentioned in the name, it contains three elements.

These elements and their relationships are described in Figure D.4.

The Model encapsulates the state and behaviour of the application. The responsability of the View is to render the Model, and the Controller processes user events and drives the Model and Views updates.

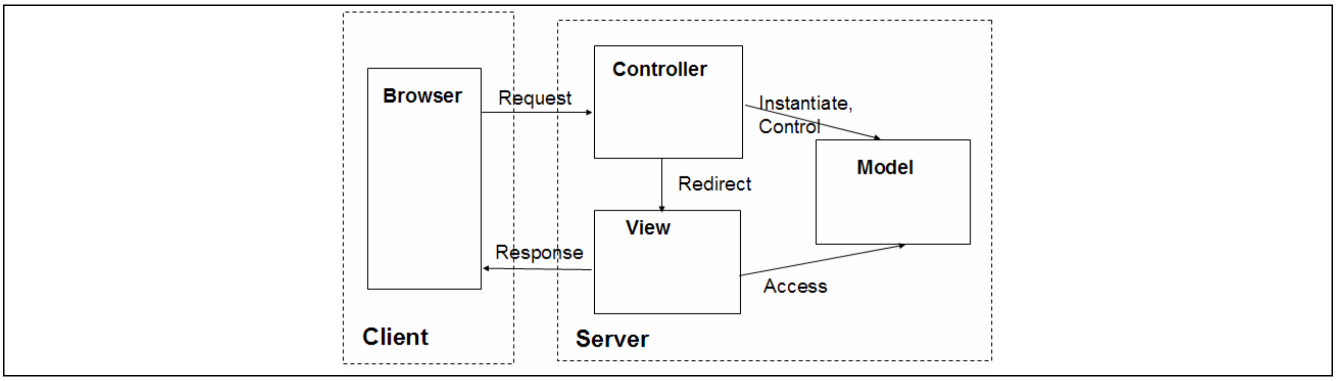



**Figure D.4 – MVC elements**

Figure D.5 shows the UML Profile that can be used to model this style in your Engineering Viewpoint.

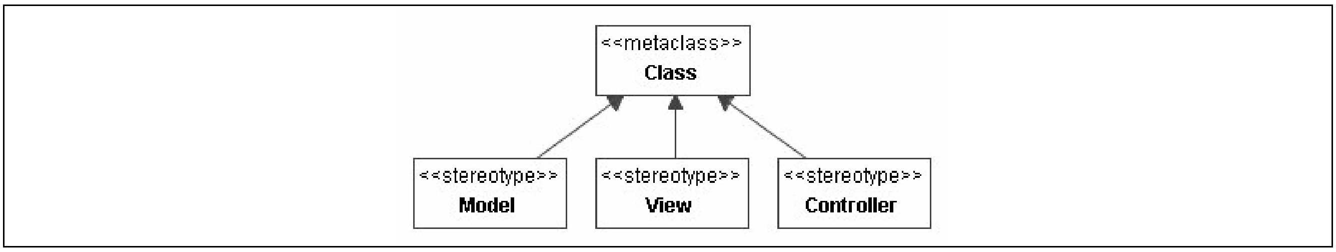



**Figure D.5 – MVC UML Profile**

### D.2.4 Service-Oriented Architecture (SOA)

The use of heterogeneous technologies and applications and the need to integrate them together is now a reality for any company. The purpose of SOA is to make that integration easier by providing, through services, a loosely coupling between those different applications.

A Service is a logical component that defines a set of interfaces and that is not allocated to a defined user but to multiple clients which can share it. A Service Provider is a component that implements the service interfaces. Services and service providers are published and accessed via a repository called a Service Registry. These services can be discovered and accessed by consumers (end-user applications or other services) through the service registry, following the principles shown in the figure below.

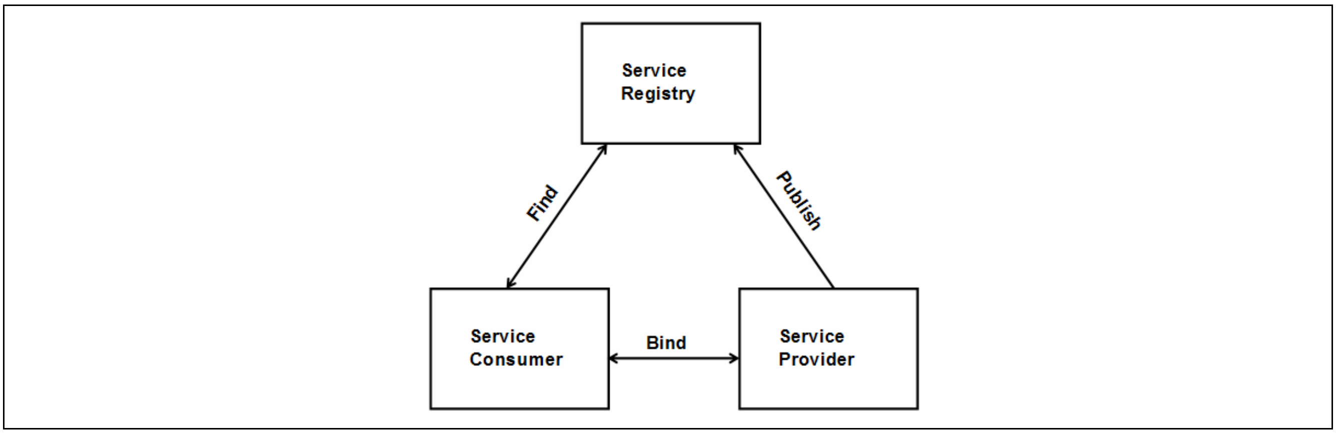



**Figure D.6 – SOA**

The corresponding UML Profile is shown in Figure D.7.

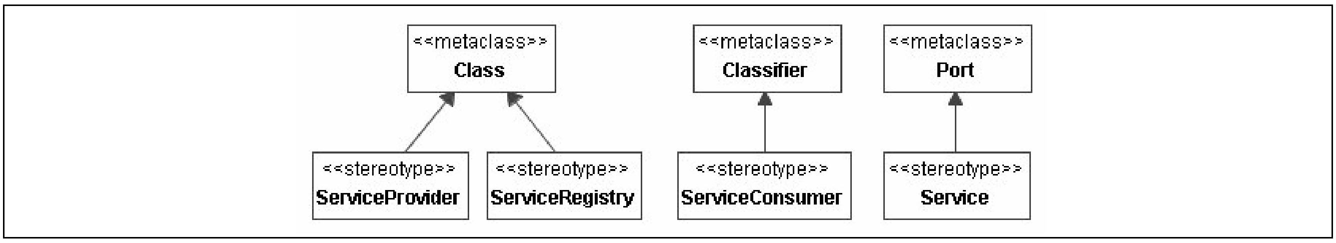



**Figure D.7 – SOA UML Profile**

# Bibliography

This annex contains a list of standards, projects and initiatives related to the contents of this document, and a list of books, articles, and research papers that are related to the use of UML for specifying ODP systems or any of the ODP viewpoints.

This annex is not normative.

Temporary Note – National bodies are invited to contribute to these lists by providing information, links, and references to projects and papers that can be included in this annex.

## Related standards

- omg/03-06-01, MDA Guide
- ptc/02-02-05, UML profile for Enterprise Distributed Object Computing

## Related projects and initiatives

- Synapses project. (http://www.cs.tcd.ie/synapses/public/)
- INTAP (http://www.net.intap.or.jp/e)
- COMBINE project (http://www.opengroup.org/combine/overview.htm)
- NASA’s Reference Architecture for Space Data Systems (RASDS)
- EDF’s DASIBAO project.

## References


[1] Peter Linington, “Options for Expressing ODP Enterprise Communities and their Policies by Using UML.” In *Proceedings of the 3rd International Enterprise Distributed Object Computing Conference* (EDOC'99), Germany, pp. 72-82, September 1999. IEEE Computer Society Press.

[2] Howard Bowman and John Derrick, “Viewpoint Modelling.” In *Formal Methods for Distributed Processing. A Survey of Object-Oriented Approaches*, Cambridge University Press. Chapter 20, pp. 451-475, 2001. Cambridge, England.

[3] Maarten W. Steen and John Derrick, “ODP Enterprise Viewpoint Specification.” In *Computer Standards & Interfaces*, 22(3):165-189, September 2000. Elsevier.

[4] Francisco Durán and Antonio Vallecillo, “Formalizing ODP Enterprise Specifications in Maude.” *Computer Standards & Interfaces*, 25(2):83-102, June 2003. Elsevier.

[5] Jan Aagedal and Zoran Milosevic, “ODP Enterprise Language: UML Perspective.” In *Proceedings of the 3rd International Enterprise Distributed Object Computing Conference* (EDOC'99), Germany, pp. 60-71, September 1999. IEEE Computer Society Press.

[6] Xavier Blanc and Marie-Pierre Gervais and Raymonde Le Delliou, “Using the UML Language to Express the ODP Enterprise Concepts.” In *Proceedings of the 3rd International Enterprise Distributed Object Computing Conference* (EDOC'99), Germany, pp. 50-59, September 1999. IEEE Computer Society Press.

[7] Anne Picault, Philippe Bedu, Juliette Le Delliou, Jean Perrin and Bruno Traverson “Specifying an Information System Architecture with DASIBAO.” In *Proceedings of the International Conference on Enterprise Information Systems* (ICEIS 2004), Portugal, September 2004.

[8] F. Durán, M. Roldán, A. Vallecillo, “Using Maude to Write and Execute ODP Information Viewpoint Specifications.” *Computer Standard & Interfaces*, 27(6):597-620, 2005.

[9] OMG. “Relationship of the Unified Modelling language to the Reference Model of Open Distributed Processing.” OMG document ormsc/2001-01-01, Version 1.4, January 2001.

[10] INTAP (Interoperability Technology Association for Information Processing), Japan. “A Guide for Using RM-ODP and UML Profile for EDOC.” Document available at http://www.net.intap.or.jp/e/odp/intap-guide.pdf.

[11] Stuart Kent, “The Unified Modelling Language.” In *Formal Methods for Distributed Processing. A Survey of Object-Oriented Approaches*, Cambridge University Press. Chapter 7, pp. 126-152, 2001. Cambridge, England.

[12] D. H. Akehurst, J. Derrick, and A. G. Waters, “Addressing computational viewpoint design.” In *Proceedings of the 7th International Enterprise Distributed Object Computing Conference (EDOC 2003)*, pages 147–159, Brisbane, Australia, Sept. 2003. IEEE CS Press.

[13] E. Najm and J.-B. Stefani, "Computational models for open distributed systems." In H. Bowman and J. Derrick, editors, *Proceedings of FMOODS'97*, pages 157–176, Canterbury, 1997. Chapman & Hall.

[14] R. Sinnot and K. J. Turner, "Specifying ODP computational objects in Z." In E. Najm and J.-B. Stefani, editors, *Proceedings of FMOODS'96*, pages 375–390, Canterbury, 1996. Chapman & Hall.

[15] A. Février, E. Najm, and J.-B. Stefani, "Contracts for ODP." In *Proceedings of the 4th AMAST Workshop on Real-Time Systems, Concurrent and Distributed Software*, Mallorca, Spain, May 1997.

[16] D. Hashimoto, H. Miyazaki, A. Tanaka, "UML 2 Models for ODP Engineering/Technology Viewpoints." In Proc. of the *Second International Workshop on ODP for Enterprise Computing (WODPEC 2005)*, pages 32-38, Enschede, The Netherlands, September 2005.

[17] R. Romero, A. Vallecillo, "Modelling the ODP Computational Viewpoint with UML 2.0." In Proc. of the *9th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2005)*, pages 212-233, Enschede, The Netherlands, September 2005.

# Index

The number associated with the index entry indicates the clause or subclause where the index entry can be found.