ISO/IEC JTC1/SC7/WG17 W17N0222

ITU-T X.911 ISO/IEC 15414

Date: 05-10-06

Proposed New Version

Anaheim 2004 Output

6 February 2004

ISO/IEC JTC 1/SC 7

Software Engineering

Secretariat: Canada (SCC)

Doc Type:	. . . . .
Title:	Information Technology—Open Distributed Processing— Reference Model—Enterprise Language
	ISO/IEC 15414 | ITU-T Recommendation X.911
Source:	Project Editor
Project:	1.07.77
Status:	English-language proposed new edition; output from 2004 Anaheim
Action:	For ballot
Distribution:	SC 7 and ITU
Medium:	E
Number of Pages:	49
Version:	43
Address reply to:	joaquin@acm.org

ISO/IEC JTC1/SC7 Secretariat
École de technologie supérieure
1100, rue Notre-Dame Ouest
Montréal, Québec
Canada H3C 1K3

Tel. +1 514 396 8632
Fax. +1 514 396 8684

INTERNATIONAL STANDARD
ISO/IEC 15414 : Draft International Standard
ITU-T Rec. X.911 (Draft Recommendation)
ITU-T RECOMMENDATION

INTERNATIONAL TELECOMMUNICATION UNION

ITU-T **X.911**

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(05/2005)

COMMON TEXT DRAFT PUBLICATION

Information Technology—Open Distributed Processing—Reference Model—Enterprise Language

Proposed new edition

ITU-T Recommendation X.911

International Standard ISO/IEC 15414
ITU-T Recommendation X.911

Information technology – Open distributed processing – Reference model – Enterprise language

Summary

This Recommendation | International Standard provides:

a) a language (the enterprise language) comprising concepts, structures, and rules for developing, representing, and reasoning about a specification of an Open Distributed Processing (ODP) system from the enterprise viewpoint (as defined in ITU-T Rec. X.903 | ISO/IEC 10746-3);

b) rules which establish correspondences between the enterprise language and the other viewpoint languages (defined in ITU-T Rec. X.903 | ISO/IEC 10746-3) to ensure the overall consistency of a specification.

Contents

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 standardization 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.

The Reference Model of Open Distributed Processing 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.

This Recommendation | International Standard refines and extends the definition of how ODP systems are specified from the enterprise viewpoint, and is intended for the development or use of enterprise specifications of ODP systems.

0.1 RM-ODP

The RM-ODP consists of:

– Part 1: ITU-T Rec. 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 categorization of required areas of standardization expressed in terms of the reference points for conformance identified in ITU-T Rec. X.903 | ISO/IEC 10746-3. This part is not normative.

– Part 2: ITU-T Rec. X.902 | ISO/IEC 10746-2: **Foundations**: which contains the definition of the concepts and analytical framework for normalized 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 Rec. X.903 | ISO/IEC 10746-3 and to establish requirements for new specification techniques. This part is normative.

– Part 3: ITU-T Rec. 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 Rec. X.902 | ISO/IEC 10746-2. This part is normative.

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

– ITU-T Rec. X.911 | ISO/IEC 15414: **Enterprise language**: this Recommendation | International Standard.

0.2 Overview and motivation

Part 3 of the Reference Model, ITU-T Rec. X.903 | ISO/IEC 10746-3, defines a framework for the specification of ODP systems comprising:

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

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

The purpose of this Recommendation | International Standard is to:

– Refine and extend the enterprise language defined in ITU-T Rec. X.903 |ISO/IEC 10746-3 to enable full enterprise viewpoint specification of an ODP system;

– Explain the correspondences of an enterprise viewpoint specification of an ODP system to other viewpoint specifications of that system; and

– Ensure that the enterprise language when used together with the other viewpoint languages is suitable for the specification of a concrete application architecture to fill a specific business need.

This Recommendation | International Standard uses concepts taken from ITU-T Recommendations X.902 and X.903 | ISO/IEC 10746-2 and 10746-3 and structuring rules taken from clause 5 of ITU-T Rec. X.903 | ISO/IEC 10746-3; it introduces refinements of those concepts, additional viewpoint-specific concepts, and prescriptive structuring rules for enterprise viewpoint specifications. The additional viewpoint-specific concepts are defined using concepts from ITU-T Recommendations X.902 and X.903 | ISO/IEC 10746-2 and 10746-3.

This Recommendation | International Standard provides a common language (set of terms and structuring rules) to be used in the preparation of an enterprise specification capturing the purpose, scope and policies for an ODP system. An enterprise specification is a part of the specification of an ODP system using viewpoints defined by ITU-T Recommendation X.903 | ISO/IEC 10746-3. The specification of the ODP system can describe any or all of:

– an existing system within its environment;

– an anticipated future structure or behaviour of that existing system within the same or an anticipated future environment;

– a system to be created within some environment.

The primary audience for this Recommendation | International Standard is those who prepare and use such specifications. The audience includes ODP system owners and users, including subject manager experts, and developers and maintainers of ODP system, tools, and methodologies.

The motivation for the enterprise language is to support standardised techniques for specification. This improves communication and helps create consistent specifications.

The preparation of specifications often falls into the category referred to as analysis or requirement specification. There are many approaches used for understanding, agreeing and specifying systems in the context of the organisations of which they form a part. The approaches can provide useful insights into both the organisation under consideration and the requirements for systems to support it, but they generally lack the rigour, consistency and completeness needed for thorough specification. The audiences of the specifications also vary. For agreement between the potential users of an ODP system and the provider of that system, it may be necessary to have different presentations of the same system—one in terms understood by clients, and one in terms directly related to system realization.

The use of enterprise specifications can be wider than the early phases of software engineering process. A current trend is to integrate existing systems into global networks, where the functionality of interest spans multiple organisations. The enterprise language provides a means to specify the joint agreement of common behaviour of the ODP systems within and between these organisations. The enterprise specification can also be used at other phases of the system life cycle. The specification can, for example, be used at system run-time to control agreements between the system and its users, and to establish new agreements according to the same contract structure. Enterprise viewpoint specifications may contain rules for inter-organisational behaviour.

This standard also provides a framework for development of software engineering methodologies and tools exploiting ODP viewpoint languages, and a set of concepts for development of enterprise viewpoint specification languages. For these purposes this standard provides rules for the information content of specifications and the grouping of that information. Further requirements on the relationships between enterprise language concepts and concepts in other viewpoints are specific to the methodologies, tools or specification languages to be developed.

An enterprise specification defines the purpose, scope, and policies of an ODP system and it provides a statement of conformance for system implementations. The purpose of the system is defined by the specified behaviour of the system while policies capture further restriction of the behaviour between the system and its environment or within the system itself related to the business decisions of the system owners.

An enterprise specification also allows the specification of an ODP system that spans multiple domains and is not owned by a single party, and specification of the collective behaviour of a system that is divided into independently specified and independently working subsystems.

Annex A present parts of a model of the enterprise language, illustrating the concepts of the enterprise language and their relationships. Annex B explains concepts and structuring rules of the enterprise language and provides examples of how they may be used. These annexes are not normative.

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION

Information technology – Open distributed processing – Reference model – Enterprise language

1 Scope

This Recommendation | International Standard provides:

a) a language (the enterprise language) comprising concepts, structures, and rules for developing, representing, and reasoning about a specification of an ODP system from the enterprise viewpoint (as defined in ITU-T Rec. X.903 | ISO/IEC 10746-3);

b) rules which establish correspondences between the enterprise language and the other viewpoint languages (defined in ITU-T Rec. X.903 | ISO/IEC 10746-3) to ensure the overall consistency of a specification.

The language is specified to a level of detail sufficient to enable the determination of the compliance of any modelling language to this Recommendation | International Standard and to establish requirements for new specification techniques.

This Recommendation | International Standard is a refinement and extension of ITU-T Rec. X.903 | ISO/IEC 10746-3, clauses 5 and 10, but does not replace them.

This Recommendation | International Standard is intended for use in preparing enterprise viewpoint specifications of ODP systems, and in developing notations and tools to support such specifications.

As specified in clause 5 of ITU-T Rec. X.903 | ISO/IEC 10746-3, an enterprise viewpoint specification defines the purpose, scope and policies of an ODP system. [see also 3-5.0]

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 the currently valid ITU-T Recommendations.

Identical ITU-T 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*.

3 Definitions

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 Rec. X.902 | ISO/IEC 10746-2.

– action;

– activity;

- behaviour (of an object);
- component object [2-5.1];
- composite object;
- composition;
- configuration (of objects);
- conformance;
- conformance point;
- contract;
- <X> domain;
- entity;
- environment contract;
- environment (of an object);
- epoch;
- establishing behaviour;
- instantiation (of an <X> template);
- internal action;
- invariant;
- liaison;
- location in time;
- name;
- object;
- obligation;
- ODP standards;
- ODP system;
- permission;
- prohibition;
- proposition;
- reference point;
- refinement;
- role;
- state (of an object);
- subsystem [2-6.5];
- subtype;
- system;
- <X> template;
- terminating behaviour;
- 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 Rec. 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 ODP standards extended in this specification

This Recommendation | International Standard extends the definition of the following term originally defined in ITU-T Rec. X.902 | ISO/IEC 10746-2. [2-11.2.7]:

- policy.

The extended definition is in clause 6.

4 Abbreviations

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

ODP	Open distributed processing
RM-ODP	Reference Model of Open Distributed Processing (ITU-T Recommendations X.901 to X.904 | ISO/IEC 10746 Parts 1-4)

5 Conventions

This Recommendation | International Standard contains references to Parts 2 and 3 of the RM-ODP and to the normative text of this Recommendation | International Standard. Each reference is of one of these forms:

[2-n.n] — a reference to clause n.n of RM-ODP Part 2: Foundations, X.902 | IS 10746-2;

[3-n.n] — a reference to clause n.n of RM-ODP Part 3: Architecture, X.903 | IS 10746-3.

[n.n] — a reference to clause n.n of this Recommendation | International Standard;

For example, [2-9.4] is a reference to Part 2 of the reference model, (ITU-T Rec. X.902 | ISO/IEC 10746-2), subclause 9.4 and [6.5] is a reference to clause 6.5 of this Recommendation | International Standard. These references are for the convenience of the reader.

This Recommendation | International Standard also contains some text which is a modification of text from Part 3 of the reference model, ITU-T Rec. X.903 | ISO/IEC 10746-3. Such text is marked by a reference like this: [see also 3-5.n]. The modifications are authoritative with respect to the enterprise language.

6 Concepts

The concepts of the enterprise language defined in this document comprise:

– the concepts identified in 3.1.1 and 3.1.2 as they are defined in ITU-T Rec. X.902 | ISO/IEC 10746-2 and in ITU-T X.903 | ISO/IEC 10746-3;

– the concepts defined in this clause.

This clause defines new concepts and refines the definition of policy from ITU-T Rec. X.902 | ISO/IEC 10746-2. [2-11.2.7] The grouping into subclauses and the headings of the subclauses of this clause are not normative.

6.1 System concepts

6.1.1 scope (of a system): The behaviour that system is expected to exhibit.

6.1.2 field of application (of a specification): The properties the environment of the ODP system shall have for the specification of that system to be used.

6.2 Community concepts

6.2.1 objective (of an <X>): Practical advantage or intended effect, expressed as preferences about future states.

NOTE 1 – Some objectives are ongoing, some are achieved once met.

NOTE 2 – In the text of ITU-T Rec. X.903 | ISO/IEC 10746-3 [3-5] the terms, purpose and objective, are synonymous. The enterprise language emphasizes the term, objective, and emphasizes the need of expressing an objective in measurable terms.

6.2.2 community object: A composite enterprise object that represents a community. Components of a community object are objects of the community represented.

6.3 Behaviour concepts

6.3.1 actor (with respect to an action): A role (with respect to that action) in which the enterprise object fulfilling the role participates in the action. That object may be called an actor.

NOTE – It may be of interest to specify which actor initiates that action.

6.3.2 artefact (with respect to an action): A role in (with respect to that action) which the enterprise object fulfilling the role is referenced in the action. That object may be called an artefact.

NOTE – An enterprise object that is an artefact in one action can be an actor in another action.

6.3.3 resource (with respect to an action): A role (with respect to that action) in which the enterprise object fulfilling the role is essential to the action, requires allocation, or may become unavailable. That object may be called a resource.

NOTE 1 – Allocation of a resource object may constrain other behaviours for which that resource is essential.

NOTE 2 – A consumable resource object may become unavailable after some amount of use. Any resource object may become unavailable after some amount of time (for example, in case a duration or expiry has been specified for the resource).

6.3.4 interface role: A role in a community, identifying behaviour which takes place with the participation of objects that are not members of that community.

6.3.5 process: A collection of steps taking place in a prescribed manner and leading to an objective.

NOTE 1 – A process may have multiple starting points and multiple end points.

NOTE 2 – The prescribed manner may be a partially ordered sequence.

NOTE 3 – A process specification can be a workflow specification.

NOTE 4 – The activity structure concepts provided in subclause 13.1 of ITU-T Rec. X.902 | ISO/IEC 10746-2 may be used, after substitution of 'step' for 'action' and 'process' for 'activity', to specify the structure of a process.

NOTE 5 – An enterprise specification may define types of processes and may define process templates.

6.3.6 **step**: An abstraction of an action, used in a process, that may leave unspecified objects that participate in that action.

6.4 Policy concepts

6.4.1 **policy**: A set of rules related to a particular purpose. A rule can be expressed as an obligation, an authorization, a permission or a prohibition. [see also 2-11.2.7]

NOTE 1 – Not every policy is a constraint. Some policies represent an empowerment.

NOTE 2 – This definition refines subclause 11.2.7 ITU-T Rec. X.902 | ISO/IEC 10746-2, by adding authorization.

6.4.2 **authorization**: A prescription that a particular behaviour shall not be prevented.

NOTE – Unlike a permission, an authorization is an empowerment.

6.4.3 **violation**: An behaviour contrary to that required by a rule.

NOTE – A rule or policy may provide behaviour to occur upon violation of that or some other rule or policy.

6.5 Accountability concepts

6.5.1 **party**: 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.

NOTE 1 – Examples of parties include enterprise objects representing natural persons, legal entities, governments and their parts, and other associations or groups of natural persons.

NOTE 2 – Parties are responsible for their actions and the actions of their agents.

The following concepts are used to identify actions which involve the accountability of a party.

6.5.2 **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.

NOTE – The enterprise object(s) participating in an action of commitment may be parties or agents acting on behalf of a party or parties. In the case of an action of commitment by an agent, the principal becomes obligated.

6.5.3 **declaration**: An action that establishes a state of affairs in the environment of the object making the declaration.

NOTE – The essence of a declaration is that, by virtue of the act of declaration itself and the authority of the object or its principal, it causes a state of affairs to come into existence outside the object making the declaration.

6.5.4 **delegation**: The action that assigns authority, responsibility or a function to another object.

NOTE – A delegation, once made, may later be withdrawn.

6.5.5 **evaluation**: An action that assesses the value of something.

NOTE 1 – For example, the action by which an ODP system assigns a relative status to some thing, according to estimation by the system.

NOTE 2 – Value can be considered in terms of usefulness, importance, preference, acceptability, etc.; the evaluated target may be, for example, a credit rating, a system state, a potential behaviour, etc.

6.5.6 **prescription**: An action that establishes a rule.

6.5.7 **agent**: An enterprise object that has been delegated (authority, responsibility, a function, etc.) by and acts for a party (in exercising the authority, carrying out the responsibility, performing the function, etc.).

NOTE 1 – An agent may be a party or may be the ODP system or one of its components. Another system in the environment of the ODP system may also be an agent of some party.

NOTE 2 – The delegation may have been direct, by a party, or indirect, by an agent of the party having authorization from the party to so delegate.

6.5.8 **principal**: A party that has delegated (authority, a function, etc.) to another.

7 Structuring rules

This clause refines and extends the structuring rules defined in subclause 5.2 of ITU-T Rec. X.903 | ISO/IEC 10746-3, as they apply to the concepts of community, enterprise object, objective, behaviour and policy. It defines structuring rules for the accountability concepts defined in 6.5. It uses the concepts defined in ITU-T Rec. X.902 | ISO/IEC 10746-2, in subclause 5.1 of ITU-T Rec. X.903 | ISO/IEC 10746-3 and in clause 6.

7.1 Overall structure of an enterprise specification

An enterprise specification of an ODP system is a description of that system and relevant parts of its environment. The enterprise specification focuses on the scope and purpose of that system and the policies that apply to it in the context of its environment.

NOTE 1 – The environment of an ODP system and the ODP system itself may span multiple organizations. More than one party may own the ODP system.

NOTE 2 – An enterprise specification may specify the collective behaviour of separately specified and inter-working subsystems of the ODP system.

A fundamental structuring concept for enterprise specifications is that of community. A community is a configuration of enterprise objects that describes a collection of entities (e.g. human beings, information processing systems, resources of various kinds and collections of these) that is formed to meet an objective. These entities are subject to an agreement governing their collective behaviour. The assignment of actions to the enterprise objects that comprise a community is defined in terms of roles. (See 7.8.1 and 7.8.2.)

The enterprise specification includes, within the areas of interest of the specification users, the objective and scope of the ODP system, the policies for the ODP system (including those of any environment contracts), the community in which ODP system is specified and the roles fulfilled by the ODP system and other enterprise objects in that community, and the processes in which the ODP system and enterprise objects in its environment participate.

An enterprise specification of an ODP system includes at least the community in which that system may be represented as a single enterprise object interacting with its environment. Whether the specification actually includes that level of abstraction is left for the specifier to decide.

NOTE 2 – This minimal enterprise specification describes the objective and scope of the ODP system; this description is necessary for completeness of the enterprise specification.

Where necessary for clarity or completeness of description of the behaviour of the ODP system, the enterprise specification can include any other communities of which the ODP system or its components are members, and other communities of which enterprise objects in the environment of the ODP system are members.

NOTE 3 – The set of communities in an enterprise specification may include, for example, communities at both more abstract and more detailed levels than the minimal enterprise specification, as well as communities relating to functional decomposition of the ODP system and to ownership of the ODP system and its parts.

The enterprise specification can also be structured in terms of a number of communities interacting with each other.

NOTE 4 – This may be, for example, a federation.

The scope of the system is defined in terms of its intended behaviour; in the enterprise language this is expressed in terms of roles or processes or both, policies, and the relationships of these.

NOTE 5 – It may be meaningful to discuss the intended, delivered or expected scope of a system in various phases of planning, development or deployment. In such cases, the term "scope" should be appropriately qualified.

A complete ODP system specification indicates rules for internal consistency in terms of relationships between various viewpoint specifications and a complete enterprise specification contains conformance rules that define the required behaviour of the described ODP system

This clause defines how the concepts identified in clause 3 or defined in clause 6 are used in an enterprise specification.

7.2 Contents of an enterprise specification

An enterprise specification is structured in terms of the elements explained in 7.1 and the other concepts identified in clause 6, as well as the relationships between them.

For each of these elements, depending on the specifier's choice and desired level of detail, the enterprise specification provides:

– the characteristics of the element; or

– the type or types of the element; or

– a template for the element.

An enterprise specification provides a pattern for realization of an ODP system in its environment. As such it may be realized once, never, or many times, depending upon the objective of the specifier. This means that the behaviour defined may also be observable any number of times, depending on when and where the specification is realized. It is therefore necessary to take care of the context when interpreting statements about the occurrence of the concepts in an enterprise specification.

In particular, when distinguishing type and occurrence in a specification, the objective is normally to distinguish between multiple occurrences of a single type within the specification, and not to imply a constraint on how often the specification can be realized in the world. The definitions in this document should be interpreted in the context of specification, without constraining when and where the specification should be realized.

The enterprise language makes no prescription about the specification process nor about the level of abstraction to be used in an enterprise specification.

NOTE 1 – No recommendations are made about the relative merits of modelling from top-down or bottom-up. Nor is there a recommended sequencing of the development of viewpoint specifications.

NOTE 2 – It is a design choice whether a specification deals with a specific implementation by, for example, identifying individual enterprise objects, or deals with a more flexible architecture by identifying types and rules for assigning enterprise objects to roles.

NOTE 3 – A specification may be partitioned because of readability, reuse of specification fragments in other specifications or interoperability of enterprise objects.

NOTE 4 – Roles and communities, as well as types and templates, can be private to a specification and development environment, or they can be stored in a repository that can be shared by a wider audience of several development environments and groups.

7.3 Community rules

7.3.1 Community

An enterprise specification states the objective of a community, how it is structured, what it does, and what objects comprise it. The objective of the community is expressed in a contract that specifies how the objective can be met. This contract:

- states the objective for which the community exists;
- governs the structure, the behaviour and the policies of the community;
- constrains the behaviour of the members of the community;
- states the rules for the assignment of enterprise objects to roles.

The contract of the community specifies constraints that govern the existence or behaviour of the collection of entities described by the community. When a collection of entities is represented as a community, there may already be some implicit or explicit agreement about those entities. Terms of that agreement may appear in the contract of the community. An enterprise specification may include all or part of that agreement by reference. Such references relate the elements of the specification to terms of that agreement. In particular, commitments of enterprise objects may be subject to that agreement.

The behaviour of the community is such that it meets its objective. The enterprise objects of a community are constrained by the rules of the contract of the community.

The contract can be put in place by a defined behaviour carried out by enterprise objects or the contract may be prescribed to exist by the enterprise specification.

The collective behaviour of the community is specified in terms of one or more of the following elements:

- the roles of the community (including those roles which define how a community interacts with its environment);
- the processes that take place in the community;
- the assignment of roles to steps in processes;
- policies that apply to the roles and processes; and
- identification of those actions for which parties are accountable.

This collective behaviour is constrained by the policies associated with roles and processes and by the contract of the community.

The behaviours of objects in a community are subject to the contract of that community and to the constraints specified in relationships between those objects.

The community is further defined in terms of the following elements:

- roles;
- policies for assignment of enterprise objects to roles;
- relationships between roles;
- relationships of roles to processes;

- policies that apply to roles and to relationships between roles;
- policies that apply to relationships between enterprise objects in the community;
- behaviour that changes the structure or the members of the community during the lifetime of that community.

NOTE 1 – Types of communities or a community template may be used in the specification of a community.

NOTE 2 – Types of communities may be related by refinement.

NOTE 3 – A family of related contracts may be generated from a contract template. Some aspects of the contract (e.g. membership) may only apply to particular instantiations of the contract template, while other aspects may apply to all instantiations of the contract template. For example, assignment rules and policies can be considered as parameters in a contract template. The style of contract specification determines the method of community establishment, as well as other aspects of the community life-cycle.

NOTE 4 – The specification of a community may include specific enterprise objects, relationships between those objects, and relationships of those object to enterprise objects assigned to roles in that community.

7.3.2 Relationships between communities

An enterprise specification can include one or more communities. Interactions between enterprise objects fulfilling appropriate roles within different communities can be considered as interactions between those communities.

Communities may interact in the following ways:

- a community object fulfils one or more roles in other communities;
- two or more community objects interact in fulfilling roles in some other community;
- the enterprise specification requires the same object to fulfil specific roles in more than one community and the behaviour of the object in any given role may affect its behaviour in other roles;
- an object, in fulfilling an interface role (see 7.8.3) of one community, interacts with an object fulfilling an interface role in another community
- a community includes behaviour for creating new communities.

NOTE 1 – For example, federation establishment means creation of a new community, which involves putting in place the contract of the community, including the structure and policies for that community.

NOTE 2 – For interactions involving community objects and the communities they represent see 7.8.3 – Interface roles and interactions between communities.

For each of these ways of interacting there is an invariant that determines the constraints on the collective behaviour of the communities concerned.

These invariants include:

- where a community object fulfils one or more roles in another community, the community that the community object represents is governed by the policies of the other community;
- where two or more community objects interact in fulfilling roles in some other community, the communities that the community objects represent are related by those interactions;
- where the same object is required to fill specific roles in more than one community, an invariant specifies how the actions of that object affect those communities;
- where the same object is required to fill specific roles in more than one community, that object becomes governed by the policies of all those communities;
- where two or more communities interact, there is a set of policies common to those communities.

NOTE 3 – Where two communities interact, an implicit community may be considered, such that the community objects representing both communities are members of and are governed by the policies of that community. The element of shared objective and the common set of policies can be formed either at design time and included in the specifications of the communities or left for run-time negotiation or testing of acceptability during community population.

NOTE 4 – The communities involved may have differing rules; enterprise object must be able to conform to all these rules.

7.4 Enterprise object rules

An enterprise specification will include enterprise objects; an enterprise object is any object in an enterprise specification. Any enterprise objects and the entities they model are those felt to be necessary or desirable to specify the system from the enterprise viewpoint or to understand the enterprise specification.

NOTE 1 – An enterprise object may be a model of a human being, a legal entity, an information processing system, a resource or a collection or part of any of these.

An enterprise object may be refined as a community at a greater level of detail. Such an object is then a community object.

All enterprise objects in an enterprise specification fulfil at least one role in at least one community. In fulfilling their roles, enterprise objects participate in actions, some of which are interactions with other enterprise objects. The behaviour of an enterprise object is restricted by the roles to which it is assigned.

An enterprise object may be a member of a community because:

– by design the community includes the object;

– the object becomes a member of the community at the time of creation of that community; or

– the object becomes a member of the community as a result of dynamic changes in the configuration of the community.

NOTE 2 – The contract of the community includes rules for the assignment of enterprise objects to roles; thus, to establish a community it is not necessary to identify the enterprise objects of that community.

NOTE 3 – The contract of the community can include rules that change the community structure (for example, the number of roles).

7.5 Common community types

Two common community types are:

– <X>-domain.

– <X>-federation.

Communities of these types can be specified so that they overlap totally or partially. These basic community types do not imply any hierarchical relationships. A specification may choose to use some or none of these community types.

7.5.1 <X>-domain community type

An <X>-domain community comprises an <X>-domain of enterprise objects in the roles of controlled objects and an enterprise object in the role of controlling object for the <X>-domain. The <X>-domain community establishes the characterizing relationship <X> between the enterprise objects in the roles of controlled objects and the enterprise object in the role of controlling object.

7.5.2 <X>-federation community type

An <X>-federation community is a community of some number of pre-existing communities cooperating to achieve a shared objective. Each member of a federation agrees by participating in the federation to be bound by the contract of the community (which may include obligations to contribute resources or to constrain behaviour) so as to pursue the shared objective. At the same time, a federation preserves the autonomy of the original participants. The specification of a federation may hide any aspects of the members not directly relevant to the shared objective; it may include defined behaviour that enables a participant to withdraw from the federation at any time.

7.6 Lifecycle of a community

7.6.1 Establishing a community

An enterprise specification can include establishing behaviour for a community.

The establishing behaviour may be implicit or explicit, but it establishes the required structures and responsibilities to maintain and control the community, for example, the contract of the community, the policies for the community, and the objects in the community. Objects of the community may need to be instantiated as a part of the establishing behaviour.

7.6.2 Assignment policy

The establishing behaviour by which a community is established includes the assignment of enterprise objects to roles. The contract of the community specifies an assignment policy, rules for choosing enterprise objects to fulfil the specified roles. The enabled behaviour is consistent with the roles.

NOTE 1 – The role/object relationship is not a type/instance relationship.

NOTE 2 – The assignment process can be late and dynamic, i.e. a role can be fulfilled by an enterprise object through a match-making process that considers, with respect to the requirements stated for the role, the interfaces and behaviour of that object, and, in the case of a community object, the policies of the community it represents.

Members of the community may be selected on demand according to the assignment policy for that community.

The rules of the assignment policy can directly identify the objects, or may use a supporting mechanism with more complex assignment rules. The rules may be based on object identifiers, relationships between objects, object capabilities, technologies, preceding commitments, object behaviour, etc.

7.6.3 Changes in a community

Changes in the structure or behaviour of a community can occur only if an enterprise specification includes behaviour that can cause such changes.

The changes to be considered here include:

- addition, change, and removal of policies or rules;
- addition, change, and removal of roles;
- addition and removal of enterprise objects;
- addition, change, and removal of processes or steps.

NOTE – Changes to a community shall maintain the overall consistency of the contract of that community.

The enterprise objects assigned to roles in the community can be changed during the lifetime of the community. As a consequence, a role can, subject to other constraints, have no enterprise object assigned to it. Still, the community is continuously responsible for the obligations placed on that role.

If an enterprise object ceases to fulfil the role to which it is assigned according to an assignment rule, that object violates the contract of the community.

7.6.4 Terminating a community

An enterprise specification can include terminating behaviour for a community.

NOTE 1 – For example, a contract of a community may provide for termination when the objective is achieved. A violation may be associated with a recovery behaviour, which may be the termination of the community.

NOTE 2 – Some communities are permanent and never terminate.

7.7 Objective rules

Every community has exactly one objective. The objective is expressed in a contract which specifies how the objective can be met.

An enterprise specification may decompose the objective of a community into sub-objectives. A sub-objective may be assigned to a collection of roles; in that case, the behaviour of the collection of roles is specified to meet the sub-objective and the sub-objective is met by the collection of objects performing the actions of the collection of roles.

The purpose of an ODP system is expressed as one or more objectives (or sub-objectives) of the community or set of communities in which the ODP system fulfils roles. If the ODP system is itself modelled as a community, then the purpose of the system is the objective of that community.

A sub-objective may be assigned to a process; in that case, the process is specified to meet the sub-objective and the sub-objective is met by the actions of objects performing the process. In this case, the sub-objective defines the state in which the process terminates.

The policies of a community restrict the community behaviour in such a way that it is possible to meet the objective. Such policies result in behaviour that suits the objective of the community.

When a community-object fulfils a role in another community, the objective of the community of which the community-object is an abstraction is consistent with any sub-objectives assigned to that role in the other community.

NOTE – An enterprise specification may provide for detection of conflicts in objectives and for resolution of those conflicts.

7.8 Behaviour rules

7.8.1 Roles and processes

The behaviour of a community is a collective behaviour consisting of the actions in which the objects of the community participate in fulfilling the roles of the community, together with a set of constraints on when these actions may occur.

NOTE 1 – There are many specification styles for expressing when actions may occur (e.g. sequencing, pre-conditions, partial ordering, etc.). The modelling language chosen for expressing an enterprise specification may impose certain styles.

The assignment of actions to the enterprise objects that comprise a community is defined in terms of roles. A role identifies an abstraction of the community behaviour. All of the actions of that role are associated with the same enterprise object in the community. Each action of the community is either part of a single role behaviour or is an interaction that is part of more than one role behaviour. Each of these abstractions is labelled as a role. The behaviour identified by that role is subject to the constraints specified in the contract and structure of the community. In contrast to the specification of actions and their ordering in terms of processes (see below), the emphasis is on the enterprise objects that participate in the particular behaviour.

Roles are used to decompose the behaviour of the community into parts that can each be performed by an enterprise object in the community. The enterprise object that performs the behaviour of a role is said to fulfil that role within the community or is said to be assigned to that role within the community.

Each action will be part of at least one role, but can be part of many roles (when the action involves an interaction).

The actions and their ordering can be defined in terms of processes. A process identifies an abstraction of the community behaviour that includes only those actions that are related to achieving some particular sub-objective within the community. Each abstraction is labelled with a process name. In contrast to the specification of actions as related to roles (see above), the emphasis is on what the behaviour achieves.

Processes decompose the behaviour of the community into steps.

NOTE 2 – The choice of using a role-based or process-based modelling approach will depend on the modelling method used and the aim of modelling. A combination of the two approaches may be used.

7.8.2 Role rules

In a contract of a community, each role stands as a placeholder for some enterprise object that exhibits the behaviour identified by the role. For each role there is an assignment rule that sets requirements for objects that may fulfil that role.

An enterprise object may fulfil several roles in one community, and may fulfil roles in several communities. An object fulfilling several roles becomes constrained simultaneously by all the behaviours identified by those roles and by the policies that apply to those roles.

NOTE 1 – If the term '<X> object' is used in an enterprise specification, where <X> is a role, it is be interpreted as meaning 'an enterprise object fulfilling the role, <X>'. Where an enterprise object fulfils multiple roles, the names can be concatenated.

At any location in time at most one enterprise object fulfils each role. The constraints of the behaviour identified by the role become constraints on the object fulfilling the role. A role may be fulfilled by different objects at different times or be unfulfilled, provided that the specification of the community so permits.

An enterprise specification may include a number of roles of the same type each fulfilled by distinct enterprise objects, possibly with a constraint on the number of roles of that type that can occur.

NOTE 2 – Examples are modelling the members of a committee and modelling the customers of a service.

An enterprise object assigned to a role shall be of a type behaviourally compatible with that role, unless the specification includes mechanisms to determine and resolve any incompatibilities. [2-9.4]

NOTE 3 – Enterprise specifications may refer to existing mechanisms for determining and resolving incompatibilities between types of objects and requirements set by roles, thus enlarging the set of objects acceptable for a given role.

An enterprise specification may allow roles to be created or deleted during the lifetime of the community. The role lifetime is contained within the community lifetime, and the period for which a particular enterprise object fulfils a given role is contained within the lifetime of that role.

NOTE 4 – The constraints of the community must be satisfied throughout its lifetime. However, these invariants may change; this may determine different epochs in this lifetime. Such changes may lead to changes in the sets of roles and in the sets of relationships between roles of the community.

An assignment policy is a set of rules of a community which govern the selection of an enterprise object to fulfil a role.

NOTE 5 – The rules define what the object to fulfil a role shall be capable of doing and not restricted from doing by earlier commitments, and what relationships to other objects are required or prohibited.

7.8.3 Interface roles and interactions between communities

One or more roles in a community may identify behaviour that includes interactions with objects outside that community; these are interface roles.

In such a case a community may be specified at two different levels of abstraction:

- as a configuration of enterprise objects, where some of these objects fulfil interface roles; and
- as a community object that is an abstraction of the community. Interactions in which that community object can participate as part of some other community are identified by the interface roles of the community which that community object represents.

The behaviour identified by an interface role may include internal actions.

7.8.4 Enterprise objects and actions

A way of categorizing the involvement of an enterprise object in an action is to consider it as having a role with respect to that action:

– The object can participate in carrying out the action; in this case it is said to fulfil an actor role or to be an actor with respect to that action.

– The object can be mentioned in the action; in this case it is said to fulfil an artefact role or to be an artefact with respect to that action.

– The object can both be essential for the action and require allocation or possibly become unavailable; in this case it is said to fulfil a resource role or to be a resource with respect to that action.

NOTE 1 – For every action there is at least one participating enterprise object. Where two or more enterprise objects participate in an action, it is an interaction. When only one enterprise object participates in an action, it may be an interaction, if the object interacts with itself. [2-8.3]

NOTE 2 – The specification of a role states the behaviour associated with that role, the policies applying to that role, the responsibilities associated with that role, and the relationships between roles. For example, for each role that specification includes descriptions of all actions and, for each action, identification of all the artefacts mentioned in the action and the resources used.

NOTE 3 – In this clause, the concept, role, is used in the context of the community in which a role is specified. Thus an object’s role is an identifier for some behaviour that the object exhibits in that community. In certain circumstances the behaviour identified is a specific action; in such cases, this is explicitly specified.

An actor in an action can also be an artefact with respect to that action. Likewise, an actor in an action can also be a resource with respect to that action (if it itself is used in the action).

When a resource is essential for some action, the action is constrained by the availability of that resource.

7.8.5 Process rules

In an enterprise specification, a process is an abstraction of the behaviour of some configuration of objects in which the identities of objects have been hidden as a result of the abstraction.

A process is a collection of steps taking place in a prescribed manner and leading to an objective. A step may be associated with multiple roles. Every step shall have one or more actors.

The process specification shall include specification of how it is initiated and how it terminates.

The collective behaviour of a community may be represented as a set of processes. This set can be seen as a more abstract process performed by a single role fulfilled by a community-object. Also, a step of a process can be further refined as a more detailed process.

7.9 Policy rules

7.9.1 The specification of a policy

A policy identifies the specification of a behaviour, or constraints on a 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. Changes in the policies of a community during its lifetime can occur only if an enterprise specification includes behaviour that can cause such changes.

Policies may apply to a community as a whole, to enterprise objects that fulfil roles in that community (regardless of which role), to roles (i.e. to all actions named by those roles), or to action types. They may also apply to the collective behaviour of a set of enterprise objects.

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 specification of policy may cover the degree to which, and the circumstances in which, there can be delegation to one enterprise object by another.

NOTE 1 – A policy is a named placeholder for a piece of behaviour used to parameterise a specification in order to facilitate response to later changes in circumstances. The behaviour of systems satisfying the specification can be modified by changing the policy value, subject to constraints associated with the policy in the original specification. In these terms, a policy is an aspect of the specification that can be changed, and a policy value is the choice in force at any particular instant. Thus one might speak of a scheduling policy with a FIFO policy value.

NOTE 2 – Policy may, for example, be used to configure generic objects to apply them in some specific situation, or to express a pervasive decision that affects many objects.

NOTE 3 – Policies may cover, for example, rules about:

-- behaviour (for example, processes);

-- qualities of service;

-- the names or types of objects with which a given object may interact;

-- the technology by means of which interactions may be performed;

NOTE 4 – Policies for a community may be composed from other policies; other communities may be subject to the same policies; policies may be specified in a community template; the template may include parameters used in establishing policies.

NOTE 5 – Policies for a community may form part of a hierarchy of policies. This may position the community within a larger environment, for example, with respect to some organizations.

NOTE 6 – Policies for a community are established when the community is specified or when it is established according to the specified establishing behaviour. The establishing behaviour may involve already established other communities or the controlling objects of <X> domain communities.

An enterprise object shall conform to all policies in each community in which it participates.

When an enterprise object of a community fulfils a role in another community, the policies of the two communities that apply to that object might conflict. Where an enterprise object is subject to policies of more than one community, the enterprise specification shall ensure that policy conflicts do not exist, or specify how policy conflicts are to be prevented or discovered and resolved, or state that policy conflicts are allowed to cause failures.

NOTE 7–Examples of how an enterprise specification may specify how conflict is to be prevented or resolved include specification of a policy governing the assignment of roles to objects where such conflict may arise, specification of a policy prescribing modification of behaviour, and specification of a mechanism for modification of conflicting policies.

Establishing an <X> federation community involves establishing a set of policies for that community. An enterprise object in the <X> federation community shall conform both to the policies of the <X> domain community to which it belongs and to the policies of the <X> federation community.

NOTE 8–In an inter-organisational environment, the policies for each domain community and for the federation community may have separate lifecycles.

NOTE 9– The mechanisms for conflict management may be supported by the specification language or the runtime environment of the systems involved, and thus not necessarily explicitly be visible in the enterprise specification.

NOTE 10– Examples of policy conflict cases are:

a. (Specification-time assurance) The specifications of the federation communities are compared and any conflicts are resolved in the specifications.

b. (Run-time prevention) Forming federations with policy conflicts is prevented by checking consistency of policies while assigning objects to roles in the federation community.

c. (Run-time discovery and resolution) The federation community includes a behaviour to resolve the conflict by changing policies.

d. (Failure handling) The specification of the federation community provides sanctions or alternative behaviour for cases where behaviour has failed because of policy conflicts.

The behaviour for changing the policy may include behaviour that changes the rules of that policy and behaviour that replaces that policy with a named different policy.

NOTE 11 – The behaviour may include constraints on changing that policy.

NOTE 12 – There may be no behaviour for changing the policy (that is, the policy is not changed during the lifetime of the community).

NOTE 13 – Behaviour to negotiate and change policies may be necessary to enable formation of an <X> federation

7.9.2 The specification of obligations, permissions, prohibitions and authorizations

The following subclauses provide a way of specifying policies:

7.9.2.1 Obligation

An obligation is defined by:

– an authority that prescribes the obligation;

– an identified behaviour that is subject to that authority;

– a role or roles involved in that behaviour that are subject to the authority;

– a subset of that behaviour that is required to occur;

– optionally, an object or objects that may fulfil the roles involved.

When the obligation applies, the enterprise objects fulfilling the roles that are subject to the authority shall engage in the required behaviour.

A standing obligation is an obligation that always applies.

7.9.2.2 Permission

A permission is defined by:

- an authority that prescribes the permission;
- an identified behaviour that is subject to that authority;
- a role or roles involved in that behaviour that are subject to the authority;
- a subset of that behaviour that is allowed to occur;
- optionally, an object or objects that may fulfil the roles involved.

When the permission applies, the enterprise objects fulfilling the roles that are subject to the authority are allowed to engage in the allowed behaviour.

NOTE 1 – There is, however, no guarantee that the action succeeds. For example, the action may have participants in other domains in which the action is prohibited.

An enterprise specification may specify that interactions between enterprise objects of a community may occur only when permission for the interaction exists. Such permissions may apply either to a particular role in the interaction or to the interaction as a whole.

NOTE 2 – In such cases, if permission required for an interaction is missing, that interaction fails and therefore the enterprise objects may fail to fulfil their roles.

7.9.2.3 Prohibition

A prohibition is defined by:

- an authority that prescribes the prohibition;
- an identified behaviour that is subject to that authority;
- a role or roles involved in that behaviour that are subject to the authority;
- a subset of that behaviour that shall not occur.

When the prohibition applies, the enterprise objects fulfilling the roles that are subject to the authority shall not engage in the prohibited behaviour.

NOTE – An enterprise specification may specify a behaviour by means of which the prohibited behaviour is prevented.

7.9.2.4 Authorization

An authorization is defined by:

- an authority that prescribes the authorization;
- an identified behaviour that is subject to that authority;
- a role or roles involved in that behaviour that are subject to the authority;
- a subset of that behaviour that is allowed to occur;
- optionally, an object or objects that may fulfil the roles involved.

When the authorization applies, the enterprise objects fulfilling the roles that are subject to the authority shall not be prevented from engaging in the authorized behaviour.

Authorizations will not necessarily be effective outside the domain controlled by the authority. In federations the effect of authorizations is determined by the contract of the federation.

7.9.3 Policy violations

Some violations are the result of defective specification or implementation of behaviour. Others are caused by inconsistent assumptions of communicating parties about policies.

NOTE – These may arise, for example, in a federation where there is not full control of the interacting objects or in other situations where an action is not considered to be essential enough to be specified with policies in detail for all possible participants of an interaction.

An enterprise specification can provide mechanisms for detecting violations and for appropriate recovery or sanction mechanisms.

A way of specifying policies is as policed and enforced, or unpoliced.

If policies are specified as policed and enforced this can be specified to be by optimistic or pessimistic means.

Pessimistic enforcement is preventative and requires the specification of mechanisms to ensure that the obligated actions occur, prohibited actions do not occur, and authorized actions are not prevented. Pessimistic enforcement is specified when trust is low (i.e. when non-compliance is expected) and the damage caused by non-compliance is potentially high, and when viable preventative mechanisms can be created and/or effective sanctions can be applied after non-compliance occurs.

Optimistic enforcement is not preventative. It requires the specification of mechanisms to detect and report or correct non-compliance. Optimistic enforcement is specified when trust is high and the potential damage due to non-compliance is low, and when viable preventative mechanisms do not exist.

7.10 Accountability rules

An enterprise specification identifies those actions that involve accountability of a party.

Parties can have intentions and are accountable for their actions. The concepts of subclause 6.5 are used to model an action that involves accountability of a party .

The enterprise specification identifies the actions of parties that an ODP system is prepared to participate in, respond to or record.

7.10.1 Delegation rules

An enterprise specification identifies the actions that any enterprise object that is not a party is prepared to participate in as an agent of a party. An enterprise specification describes the authority delegated to an enterprise object in terms of:

- the parties that have delegated authority to the system;
- the authority that each party has delegated;
- the duration and conditions of the delegation;
- provisions for additional delegation and withdrawal of delegation during the operation of the system.

By each such delegation, that enterprise object becomes an agent of the parties delegating, and the parties (collectively) become principal of that object. A principal is responsible for the acts of an object acting as its agent.

Insofar as provided in delegation by a party, an enterprise specification may specify further delegation, by an agent to another enterprise object.

7.10.2 Authority rules

For each authority delegated, an enterprise specification states the actions in which an agent may participate in exercising that authority. The authority delegated may be:

- to make a commitment; this binds the principal;
- to issue a declaration; this establishes the truth of some proposition just as if the principal had made the declaration;
- to make a prescription that establishes a rule; such a rule has the same force as if the principal had made the prescription;
- to further delegate an authority; this causes the agent delegated to have the authority.

7.10.3 Commitment rules

An enterprise specification identifies, for every commitment, the obligation created. It identifies, for every commitment made by an agent, the principal(s) obligated.

Establishing behaviour in an enterprise specification includes commitments by the objects participating in the establishing behaviour. If the establishing behaviour is implicit, it includes prescriptions that apply to the objects in the resulting liaison.

7.10.4 Declaration rules

A declaration identifies the changes that take place in the environment of an object as the result of an internal action of that object. An enterprise specification defines the conditions required for a particular declaration to be effective.

NOTE – A declaration may not be effective (cause the change in the environment of the object) until some interaction of the object such as, for example, a publication.

7.10.5 Prescription rules

An action of an enterprise object will be a prescription only when:

- that object is a party that by its nature may establish rules;
- that object is, in a previous epoch, specified to establish rules;
- that object is an agent of an object that may establish rules and is delegated authority to establish rules on behalf of that object; or
- the specification explicitly provides for those actions of that object that will be prescriptions.

An important special case of delegation is where the authorized action is a prescription; that is, when the delegation enables an enterprise object to make a prescription.

8 Compliance, completeness and field of application

8.1 Compliance

This document uses the term, compliance, to describe the relationship between two standards. One standard complies with another if it makes correct use of the ideas, vocabulary or framework defined there. This implies that, if a specification is compliant, directly or indirectly, with some other specifications, then the propositions which are true in those specifications are also true in a conformant implementation of the specification.

The term conformance is used for the relationship between some product and the specification from which it is produced. Conformance can be tested by inspecting the product produced to confirm the claim that its properties or behaviour are as required by the standard.

In ODP specifications, there is a need for the specifier to declare those points at which tests are to be performed and for the implementer to identify those points when offering the product for test. Large specifications are frequently organized into a specification framework populated by more detailed component specifications. The framework identifies a wide range of points at which observations can, in principle be made. These points are called reference points. The subset of reference points where tests of an implementation are required by the more detailed specifications are called the conformance points for that specification.

ODP systems are specified in terms of a number of viewpoints, and this gives rise to an accompanying requirement for consistency between the different viewpoint specifications. The key to consistency is the idea of correspondences between specifications; i.e. a statement that some terms or structures in one specification correspond to other terms and structures in a second specification.

8.2 Completeness

Specifications can be produced as a prelude to implementation, and generally change during implementation or to support system evolution. Specifications can also be produced to capture the properties of existing systems or components in order to facilitate their reuse. The references to the process of specification in this clause are intended to cover both these situations.

When a set of viewpoint specifications and correspondences is created for an ODP system, a succession of design choices is made, gradually reducing the number of conceivable implementations that would be consistent with the specification. This process is never absolutely complete, since there are always implementation choices and changes in circumstances in the environment that affect the system's behaviour, but there is some point in the design process when the specifier judges that the specification is sufficiently complete to reflect their purpose. At this point, the specification is said to have reached the viable stage. This is the stage in the specification process where it would be possible to produce some worthwhile implementation. This statement does not imply that the specification is, in any way, frozen.

The viable stage depends on the purpose of the specification, because there may be significant differences in the degree of completeness expected in, for example, an accounting policy applied to a range of independent machines or to an inter-organizational workflow. The viable stage will not be assessed to be the same for all possible applications of any particular specification notation.

8.3 Field of application

An enterprise specification includes a statement of the field of application that specifies the properties the environment shall have for the specification to be applicable.

The field of application determines whether a specification is appropriate in a given situation, and shall be satisfied before it makes sense to make observations of the real world and compare these with specified observable properties to test conformance to the specification.

NOTE – The provision of an accurate statement of the field of application is particularly important if reuse of the enterprise specification is expected. It allows the specifier who might incorporate the existing specification fragments to ask "is this specification for me?" before they begin to ask "what shall the system and its environment do?"

9 Enterprise language compliance

An enterprise specification compliant with this document shall use the concepts defined in clause 6 and those in subclause 5.1 of ITU-T Rec. X.903 | ISO/IEC 10746-3, as well as the concepts defined in ITU-T Rec. X.902 | ISO/IEC 10746-2, subject to the rules of clause 7 and those in ITU-T Rec. X.903 | ISO/IEC 10746-3 subclause 5.2.

Concepts from other modelling languages may also be employed. Where such concepts are employed, the specification concerned shall include or refer to definitions of each such concept, in terms of the concepts defined in clause 6, in ITU-T Rec. X.902 | ISO/IEC 10746-2, or in clause 5.1 of ITU-T Rec. X.903 | ISO/IEC 10746-3, and explanations of the relationships between such concepts and those defined in clause 6.

10 Conformance and reference points

This document defines the enterprise language, which provides a framework for a variety of notations to be used in specification. As such it creates a formal system that does not itself involve conformance (any more than, say, a programming language grammar involves conformance). However, specific notations derived from this standard will be supported by (generally automated) tools and design processes that produce and maintain enterprise specifications for systems, and the conformance of these tools and processes can be tested. This includes the generation of specifications that conform to the structural or grammatical rules of the language, and the construction of systems which, in operation, perform in a way consistent with the semantics of the language.

In general, such tools and processes manipulate not only the enterprise viewpoint specification but also manage correspondences with other viewpoint specifications, and so wider issues of conformance to complete sets of ODP specifications need to be considered.

NOTE – There are correspondences between each possible pair of viewpoint specifications, but the conformance issues involved are particularly important in this document because the policies expressed in the enterprise specification are reflected in all the other viewpoints.

In claiming conformance to an enterprise specification, the system provider shall state what observable reference points in the system are conformance points, and how observations at these points can be interpreted to correspond to enterprise concepts. With this information, a tester of the system is in a position to determine by observation whether the system behaves correctly. In ODP, conformance is based on the declaration of engineering viewpoint reference points (in clauses 5-7 of ITU-T Rec. X.903 | ISO/IEC 10746-3), and the implementer of an enterprise specification shall state correspondences to the engineering viewpoint in order to relate observations at the engineering reference points to enterprise concepts.

11 Consistency rules

This clause extends clause 10 of ITU-T X.903 | ISO/IEC 10746-3 by defining enterprise specification correspondences.

11.1 Viewpoint correspondences

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 an enterprise 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 from different ODP viewpoints are each complete statements in their respective viewpoint languages, using their own locally significant names, and so cannot be related without additional information in the form of correspondence statements. What is needed is a set of 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 viewpoint specifications, but do not form part of any one viewpoint specification. 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 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.

There are two kinds of standardization requirements relating to correspondences:

- 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 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.

The minimum requirement for consistency in a set of specifications for an ODP system is that they exhibit the correspondences defined in the Reference Model [3-10], those defined in this clause 11, and those defined within the specification itself.

NOTE – An enterprise specification may include objects that are not part of the ODP system being specified and may include the behaviour of such objects. Where this is the case, there may be no instances of concepts in other viewpoints that correspond to these objects or their behaviour.

11.2 Enterprise and information specification correspondences

11.2.1 Concepts related by correspondences

The enterprise concepts related are:

- community;
- enterprise object;
- enterprise action;
- role;
- policy.

The information concepts related are:

- information object;
- dynamic schema;
- static schema;
- invariant schema.

11.2.2 Required correspondences

There are no required correspondences.

11.2.3 Required correspondence statements

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.

11.3 Enterprise and computational specification correspondences

11.3.1 Concepts related by correspondences

The enterprise concepts related are:

- enterprise object;
- role;
- enterprise interaction;
- policy.

The computational concepts related are:

- computational object;
- computational behaviour;
- computational binding object;
- computational interface;
- operation;
- stream.

11.3.2 Required correspondences

There are no required correspondences.

11.3.3 Required correspondence statements

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 behaviors capable of carrying out an interaction of that enterprise interaction type.

11.4 Enterprise and engineering specification correspondences

11.4.1 Concepts related by correspondences

The enterprise concepts related are:

- behaviour;
- enterprise object;
- interaction;
- policy;
- role.

The engineering concepts related are:

- binder;
- capsule;

- channel;
- cluster;
- interceptor;
- node;
- nucleus;
- protocol object;
- stub.

11.4.2 Required correspondences

There are no required correspondences.

11.4.3 Required correspondence statements

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.

11.5 Enterprise and technology specification correspondence

In accordance with clause 15.5 of ITU-T Rec. X.902 | ISO/IEC 10746-2 and clause 5.3 of ITU-T Rec. X.903 | ISO/IEC 10746-3, 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 policy covering performance (e.g. response time), reliability, and security.

Annex A Model of the enterprise language concepts

This annex is not normative. It presents an illustrative model of the main concepts from the enterprise language and the relationships between those concepts. The selection of elements in this model, which is expressed in the diagrams below, is chosen to give a basic overview of the main features of the language, even though this results in some redundancy of expression from a formal modelling point of view. In the interests of simplicity, the rich web of relationships with all the supporting concepts from RM-ODP Part 2 has also been left out of the diagrams. In particular, relations with the concepts of type and template are not specifically illustrated.

Note 1 – The notation used in this model is UML [ISO/IEC IS 19501, Information Technology – Unified Modelling Language (UML)].

Note 2 – The convention for expressing an association is such that it can be read as: "Each (or an instance of) <Class X> <verb phrase> <cardinality> <Class Y>", where the verb phrase expresses the role played by an <X> in its relationship with a <Y>, and is placed, as a RoleEndName at the <X> end of the association (in accordance with the UML notation rule given in clause 3.43.2 of ISO/IEC DIS 19501). For example, in Figure A.1 the association that an ODP System has with Scope can be read as "Each ODP system has (the) expected behaviour defined in exactly one Scope". The choice of "each" or "an instance of" will depend on the cardinalities of the association.

The diagrams representing this illustrative model are presented below under four broad headings as found in clause 6 of this document, viz:

– System concepts – representing the relationships between an enterprise specification and the system that it describes;

– Community and Behaviour concepts – representing relationships between the main enterprise language concepts used in modelling the behaviour of a community;

– Policy concepts;

– Accountability concepts.

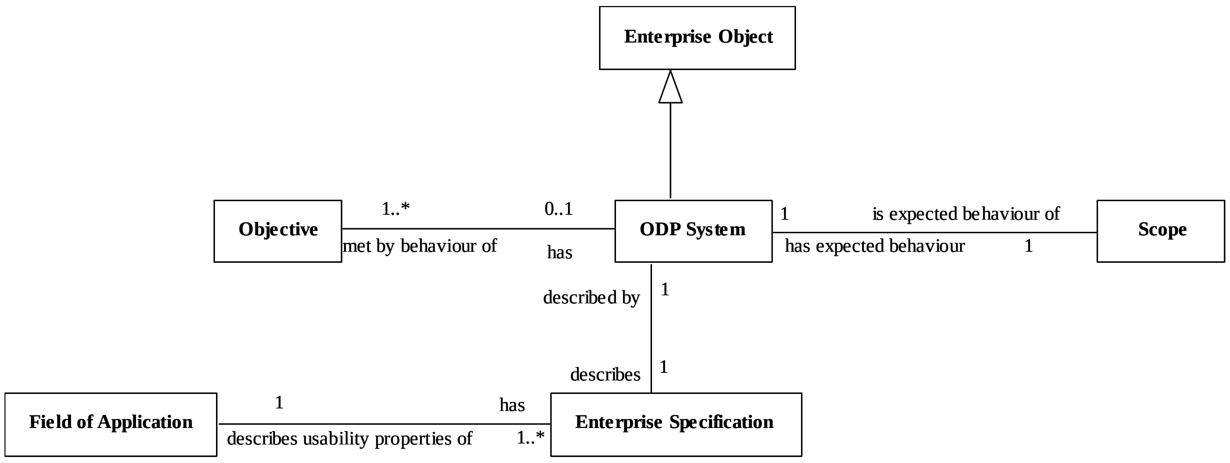

Figure A.1 System Concepts

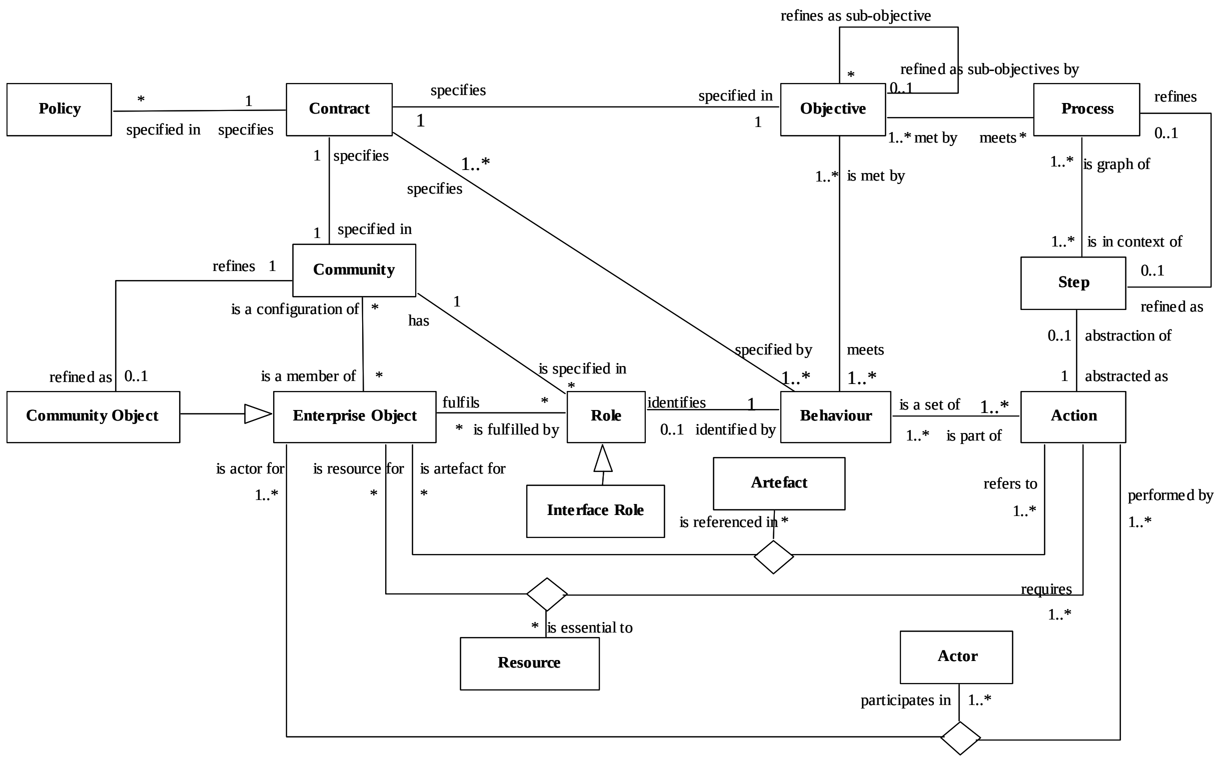

Figure A.2 Community and Behaviour Concepts

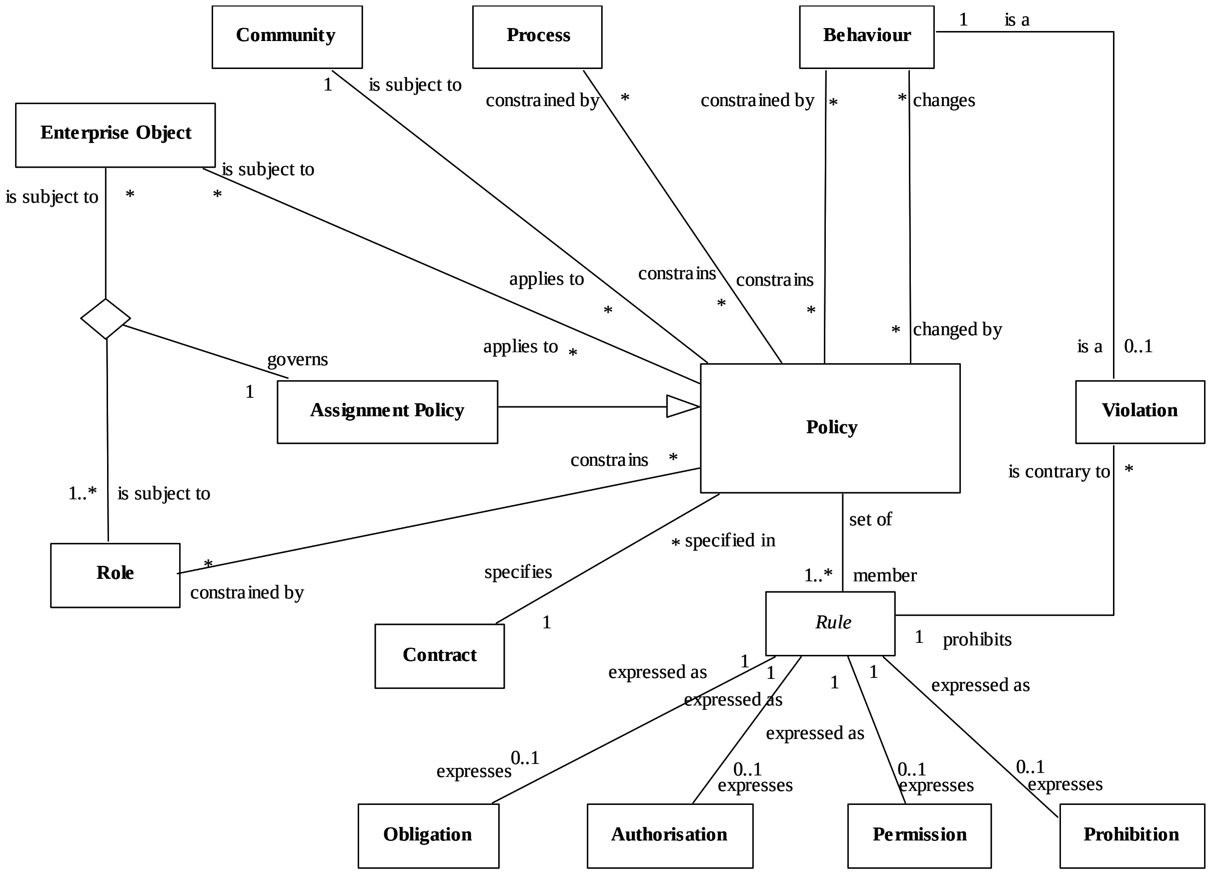

Figure A.3 Accountability Concepts

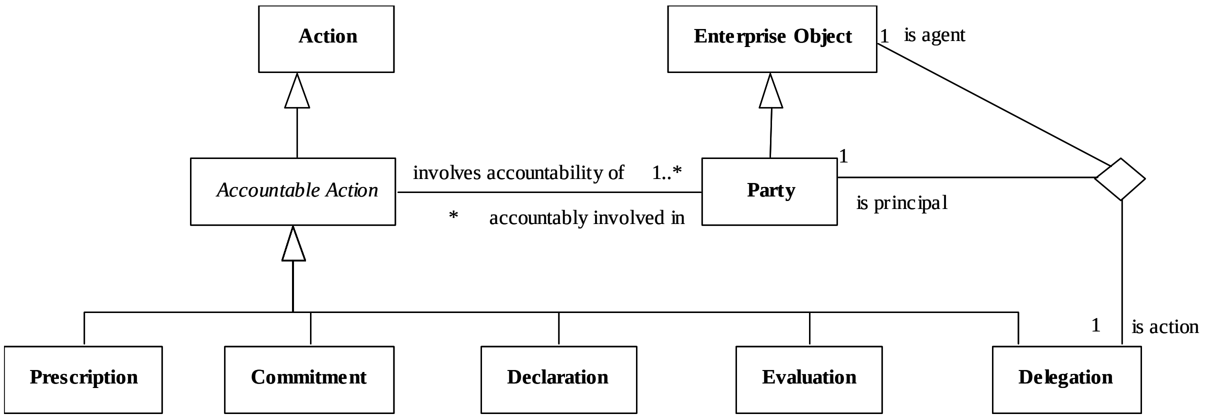

Figure A.4 Policy Concepts

Annex B Explanations and examples

This annex explains the concepts and structuring rules of the enterprise language and provides examples of how they may be used. This annex is not normative.

Each of the two parts of this annex includes a running example. The two examples illustrate (somewhat different) uses of the concepts and structuring rules to specify an ODP system from the *enterprise viewpoint*. [3-4.1.1.1]

The concepts explained are:

– Enterprise specification concepts including the specification itself, the *field of application* of the specification, the *system*, and its *scope*;

– Community concepts including, *community*, *enterprise object*, *objective*, and *contract* of the *community*;

– Behaviour concepts including *action*, *behaviour*, *role*, *process* and *step*, and *interface role*;

– Policy concepts including *rule*, *policy*, *authorization*, *obligation*, *permission*, *prohibition*, and *violation*;

– Accountability concepts including accountability, *party*, *commitment*, *declaration*, *delegation* and authority, *agent* and *principal*, *evaluation*, and *prescription*;

– More about communities, including lifecycle of a community, assignment policy, relationship between *communities*, *domains*, and *federation*.

In this annex, *terms* that refer to concepts that belong to the specification language, the RM-ODP *enterprise language*, are in *italic*, *terms* that refer to concepts that belong to universe of discourse (that is, what is being specified) [2-6], are in the usual roman typeface, and *names* used in an ODP specification are in sans serif roman. In some cases, a *term*, which refers to a concept that belongs to the enterprise language, is also occasionally used in this annex with its ordinary sense. When such a *term* is used in its ordinary sense, it is in the usual roman typeface, not in *italic*.

An ODP specification comprises one or more *viewpoint* specifications of an ODP system and its *environment*. An *enterprise viewpoint* specification is expressed in the *enterprise language*. [3-4] The *enterprise language* uses concepts taken from ITU-T Rec. X.902 | ISO/IEC 10746-2, and introduces refinements of those concepts, prescriptive rules and additional concepts defined using concepts from ITU-T Rec. X.902 | ISO/IEC 10746-2. [3-4.2.2] The additional concepts are those of clause 5 of ITU-T Rec. X.903 | ISO/IEC 10746-3 and those in this document.

B.1 First Example – Specification of an e-commerce system

This running example illustrates the use of the concepts and structuring rules to specify an *ODP system* from the *enterprise viewpoint*. [3-4.1.1.1]

Example — Our example is a specification of an e-commerce system operated by e.com, a seller of widgets. This specification is made from the *enterprise viewpoint*.

B.1.1 Enterprise Specification

B.1.1.1 Specification [3-4.2.2]

An enterprise specification of an *ODP system* focuses on the *purpose* and *scope* of that system and the *policies* for that system. That enterprise specification includes *enterprise objects* and *actions* of those objects.

An *enterprise object* is a model of, and represents, something (an *entity* [2-6.1]) in the world of the *ODP system* being specified. This may be, for example, a person or organization as well as a computer system and the software it supports.

An *action* of an *enterprise object* is a model of something that happens in the world of the ODP system being specified; the *entity* represented by that object participates in what happens. This may be, for example, something done by a person or by a computer system or one of its parts. Several *objects* may participate in the same *action*.

Example – A person places an order for a purple widget with the e.com e-commerce system. That person and the e.com e-commerce system are represented in our specification as *objects*. Placing that order for a purple widget is represented as an *action*. Those *objects* participate in that *action*.

An *action* may be a *composition*. Several *objects* participate in the *action* fulfilling the order for a purple widget. Each of those *objects* participates in some of the *actions* in a *decomposition* of that *action*.

B.1.1.2 Field of application (of a specification) [6.1.2]

To enable reuse, an *enterprise specification* will include the properties the *environment* of the *ODP system* shall have in order for that specification to be suitable for use in that *environment*.

Example – The e-commerce system specification belongs to the domains of trade and electronic business transactions, and therefore assumes the set of structures and roles common to these domains, such as buyer, seller, order, delivery, item, etc., as well as button, browser, client, server, electronic payments, etc. Moreover, the specification also assumes certain ways of doing business, and several kinds of valid operations only. Thus, the e-commerce system specification can only be applied within the domains of trade and electronic business transactions, i.e., it may not make any sense to use this specification in other environments where the aforementioned assumptions are not valid. Examples of such invalid environments include the cases of barter-based trading communities, or systems without computerized resources or with no access to electronic services (such as e-orders, e-payments, etc.).

B.1.1.3 System [2-6.5]

A *system* is something that is of interest both as a whole and as composed of parts. Some of the parts of a *system* may themselves be *systems*; these may be called *subsystems*.

Example – The e-commerce *system*, represented by an object, e-system, includes a purchasing *subsystem*, a shipping *subsystem*, and an administration *subsystem*. These are represented in the specification by three *systems*, purchasingSubsystem, shippingSubsystem, and administrationSubsystem

An *ODP system* may be an automatic information processing systems, or may be any other kind of *system*, in the general sense. Often the *ODP system* will include some parts which are computers, others which are other kinds of machinery, and parts in which people play a part.

Example — The e.com e-commerce system comprises a part that is a computer and parts that are e.com employees. The computer part handles most orders automatically, but refers some for decision-making by the e.com employees.

B.1.1.4 Scope [6.1.1]

An *enterprise specification* states the *scope* of the *ODP system*. The *scope* specifies what the system does, while suppressing details of how that is done. In any *community* in which the ODP system is represented as a single *object*, that *object* fulfils certain *roles* in that *community*; the *behaviour* identified by the *roles* fulfilled by that *object* is the *scope* of the ODP system for that *community*.

Example – The *scope* of e-system in our e-commerceCommunity includes providing a list of available kinds of widgets and the price of each, providing representations about the merchantability and fitness for use of each kind of widget, accepting orders and payments, keeping track of inventory, and so on.

B.1.2 Community

B.1.2.1 Community [3-5.1.1]

A *community* is a *configuration* of *objects* formed to meet an *objective*. That *objective* is expressed in a *contract* that specifies how the *objective* can be met. The *interactions* of *objects* fulfilling the *roles* of that *community* serve the *objective* of the *community*.

The *contract* of the community includes *policies*. These *policies* prescribe the *behaviour* of the *objects* of the *community* in situations when there are choices in *behaviour*. The same *type* of *community* can be used in different situations by setting *policies* suitable for the particular situation.

An *enterprise specification* may specify some or all of the *objects* of a *community*, or may specify or refer to supportive mechanisms that introduce *objects* into a *community* at the *creation* or *introduction* of that *community* or at other times during its lifetime. The *contract* of the community may provide for adding or removing *objects*, *roles*, or *policies* of the community.

In an *enterprise specification*, at some level of description, the *ODP system* is represented as an *enterprise object* in a *community*. That *object* may be called the ODP system object or, simply, the system. The other *objects* in that *community* are the *environment* of that *system*.

Example – In our example, the e.com e-commerce system interacts with people and with other automated systems. When the operation of that e-commerce system is specified from the *enterprise viewpoint*, that system and the context in which it operates are represented as a *community*. We will call this, e-commerceCommunity.

In the *enterprise viewpoint* specification of the *ODP system*, the e.com e-commerce system is represented as an *object* in e-commerceCommunity. That *object*, the e-commerce system object, we call e-system.

People and automated systems are also represented as *objects* in e-commerceCommunity.

The parts of the *ODP system* will be specified; these are also represented as *objects*. In some cases these parts will be under separate control or have different owners.

The e-commerce system is composed of parts. To show the *interactions* of the parts, our specification also shows e-commerceCommunity in more detail, with the e-commerce system shown as several *objects* representing parts of the e-commerce system.

The *enterprise specification* may include other *communities*. The *objects* of these *communities* will be both the parts of e-system and *objects* in the *environment* of e-system

In our specification there are also other *communities* that include e-system or some of its parts. Each of these *communities* has its own *objective*. We will see examples later in this annex.

B.1.2.2 Enterprise object [3-4.2.2]

ODP systems are modelled in terms of *objects*. An object is a model of an entity. *Enterprise objects* model the entities defined in an *enterprise specification*. [7.4]

B.1.2.3 Objective [6.2.1]

The *objective* of a *community* specifies the practical advantage or intended effect of the formation of that *community*, captured as preferences about future *states* of the *objects* of that *community*.

Example – People, firms, and automated systems interact with our example system in order to exchange goods and money. The *objective* of the e-commerceCommunity is to enable this exchange. (This *objective* may, of course, be specified in more detail.) This is captured in our *enterprise specification* as a preference that, in the future, goods will have been exchanged and everyone will be satisfied with the exchanges.

NOTE - The words '*objective*' and '*purpose*' are synonyms. The enterprise language gives a specific meaning to '*objective*,' prescribes how an *objective* is to be specified, and uses that concept throughout.

B.1.2.4 Contract [2-11.2.1]

The *contract* of the community specifies how the objective of the community can be met. That *contract* governs the *collective behaviour* of the *community*. The contract includes *policies* (sets of *rules* related to a particular purpose) that prescribe what the *objects* of the *community* may and may not do.

Example – The *contract* of our e-commerceCommunity include *policies* about privacy of customers, representations about the goods, placing of orders, means of payment, procedures in case of dissatisfaction and so on. The *policies* about means of payment will prescribe what methods of payment are permitted, how information about a method of payment will be transmitted, and the like.

This *contract* of the *community* also refers to a legal agreement between e.com and its customers. Some of the provisions of that agreement are represented in our specification as a *policy* (a set of *rules* for the purpose of ensuring that the e-commerce system complies with that legal agreement).

Essential aspects of a *contract* of a community are:

-- the *behaviour* identified by the *roles* of the *community*;

-- *policies* governing the *behaviour* of *objects* of the *community*;

-- *policies* for changing the *community* (its *roles*, *policies*, and so on); and

-- *rules* for deciding when there is a *violation* of the *contract* of the community and what *actions* are then taken.

B.1.3 Behaviour

B.1.3.1 Action [2-8.3]

An *action* in an *enterprise specification* represents something that happens in the *system* or its *environment*. An interaction takes place with the participation of more than one object.

Example – Actions in the e-commerceCommunity include *internal actions* of e-system, such as changing an inventory record and comparing the count of an item in inventory to the reorder limit for that item, and *interactions* between e-system and *objects* in its *environment*, such as requesting a price, adding a line item to an order, and requesting a payment.

B.1.3.2 Behaviour [2-8.6]

The *behaviour* of an *ODP system* is determined by the collection of all the possible *actions* in which the *system* (acting as an *object*), or any of its constituent *objects*, might take part, together with a set of constraints on when these *actions* can occur. In the *enterprise language* this is can be expressed in terms of *roles* or *processes* or both, *policies*, and the relationships of between these.

Example – *Behaviours* of the e-commerceCommunity include placing an order, shipping a widget, and collecting a payment; these all involve several *interactions* between e-system and *objects* in the environment. Internal *behaviours* include keeping track of inventory and making decisions about replenishing inventory.

B.1.3.3 Role [2-9.14]

A *role* provides the means to refer to a *behaviour*, without specifying the *object* that is associated with that *behaviour*. When the *behaviour* identified by a *role* is associated with a particular *object*, we say the *object* fulfils that *role*.

Thus a specification may state the *behaviour* of a *system* in terms of *roles*. *Roles* are used in the specification of a *community* to identify the *behaviour* of the *objects* of the *community*.

Example – The *roles* of our e-commerceCommunity include *roles* for *objects* representing the e-commerce system, customers, widget suppliers, supplier systems, and e.com managers which are *enterprise objects* in the e-commerceCommunity. The *roles* of e-system in our e-commerceCommunity include catalogueServer and orderTaker. Our specification states that the *role*, catalogueServer, includes displaying a welcome page, displaying catalogue pages, searching for kinds of widgets that meet a need described by a customer, and so on. This is one of the *roles* of e-system in our e-commerceCommunity.

A *community* specification may include *policies* for assigning *objects* to *roles*, which are called *assignment policies* (see B.1.6.2).

An *object* may fulfil several *roles* in the same *community*. On the other hand, the *contract* of the community may, for example, *prohibit* the same *object* from fulfilling both of two particular *roles*.

Example – Our specification includes the *roles*, customer and e.comManager. Because e.com is pleased to have its employees as customers, the same object, representing a manager of e.com, may fulfil both *roles*, customer and *e*.comManager.

Our specification also includes the *role*, auditor. As a matter of company policy, employees acting as auditors are not allowed to use the system as customers. So, our specification states that an *object* fulfilling the *role*, auditor, may not fulfil the role, customer.

In an enterprise specification the term, '<x> object', where '<x>' is a role, is interpreted as meaning: an *enterprise object* fulfilling the role, <x>.

Example – 'customer object' means an *object* fulfilling the *role*, customer.

B.1.3.4 Interface Role [6.3.4 and 7.8.3]

An *interface role* identifies the behaviour of an *object* of a *community* that is responsible for an *interaction* of that community with objects that do not belong to that community.

Example – In the specification, e-system is a *composite object*. Some of the *components* of e-system fulfil *roles* in the inventoryMaintenance *community*. This *community* interacts with supplierSystem objects (objects fulfilling the role, supplierSystem) outside the inventoryMaintenance *community*. The inventoryMaintenance community includes *interface roles*. The *objects* of this *community* that interact with supplierSystem objects fulfil *interface roles*.

B.1.3.5 Process [6.3.5]

A *process* is composed of *steps*. A *step* is an *action* that may leave unspecified the *objects* that participate in that *action*.

Each *step* in a *process* is associated with a *role* or *roles*. Together, *objects* fulfilling the *roles* of a *process* participate in all the *steps* of that *process*.

Example – The specification of the inventoryMaintenance *community* includes the specification of a reorderProcess. This *process* includes the *objects*, orderPlacer, receiver and inventoryKeeper and the *role*, supplier. One *step* in this *process* is the *action* in which the orderPlacer places an order. The *process* of placing an order proceeds in the same way for any supplierSystem fulfilling the role, supplier.

Part 2 of the Reference Model provides concepts for use in specifying an *activity*. These concepts may also be used to specify the structure of a *process*. [2-13.1]

Example – For handling goods as they are received, the specification of the inventoryMaintenance *community* includes the specification of a receivingProcess. This *process* includes a *forking action*. Following the fork, in one *chain* the inventoryKeeper adjusts the *inventory*; in the other *chain*, the orderPlacer adjusts outstanding *orders*. The *steps* of these two *chains* are followed by *joining action*; this is followed by *steps* completing the receivingProcess.

B.1.3.6 Enterprise objects and actions [7.8.4]

An *enterprise object*, when fulfilling a *role*, can be involved in an *action* in different ways: as an *actor* (if it participates in the *action*), as an *artefact* (if it is referenced in the *action*), or as a *resource* (if it is essential to the *action* and may become unavailable or used up).

Example – In the *action* of customer buying a purple widget, e-system object and the customer object are *actors*, the *object* representing a purple widget is an *artefact*, and shippingSubsystem is both an *actor and a resource* (represented in the *action* by a *community object* (a *composition* of the parts of that subsystem), which will be in charge of delivering the product at a later stage.)

B.1.4 Policy

B.1.4.1 Policy [2-11.2.7, 6.4]

A *policy* is a set of *rules* related to a particular purpose; a *rule* may be expressed as an *authorization*, an *obligation*, a *permission*, or a *prohibition*.

An *enterprise specification* may include *policies*, and may specify that *policies* (and *rules)* can be *prescribed* during the operation of the *ODP system*.

A *policy* may specify which of some collection of *policies* is to be applied in certain circumstances.

An *enterprise specification* may include a mechanism enabling change, from time to time, which *policy* of some collection of *policies* is to be applied in certain circumstances, or a mechanism for determining which *policy* to use in different cases of a similar situation.

Example – The specification of the e-commerce system provides for that *system* to follow two *policies* governing backorders. One *policy* is used with favoured suppliers and includes a *rule* that allows items on an order to be backordered by a supplier. This *policy* includes other *rules* governing the handling of backorders. The alternate *policy* is used with all other suppliers and includes a *rule* prohibiting backorders: if the supplier cannot fill the entire order, the order is cancelled.

B.1.4.2 Authorization [6.4.2. 7.9.2.4]

An *authorization* is a *rule* that a particular *behaviour* shall not be prevented. An *authorization* is fulfilled so long as the *authorized behaviour* is not prevented. Prevention of that *behaviour* is a *violation*.

Example – The *enterprise specification* of the e-commerce system includes a *rule*, which prescribes that an auditor object (that is, a *party* fulfilling the *role*, auditor) is *authorized* to examine any record in the system. Another *rule* prescribes that a databaseAdministrator object is *authorized* to display records when testing the operation of the dataManagementSubsystem.

B.1.4.3 Obligation [2-11.2.4]

An *obligation* is a *rule* that a particular *behaviour* is *required*. An *obligation* is fulfilled by the occurrence of the prescribed *behaviour*. If that *behaviour* does not occur as prescribed, then there is a *violation*. Some *obligations* are continuing: the *behaviour* is required to be ongoing.

Example – The *enterprise specification* includes a *rule* which prescribes, for any order accepted before 4 PM, to schedule shipment on the same day of all ordered widgets in stock and not already scheduled for shipment at the time of acceptance of that order. This *rule* is represented as an *obligation* of the shippingSubsystem.

B.1.4.4 Permission [2-11.2.5]

A *permission* is a *rule* that a particular *behaviour* is allowed to occur.

Example – The *enterprise specification* includes *rules* that certain *subsystems* may establish *communication* with *objects* outside the e.com *security domain*. These *rules* are *permissions*.

A *permission* for a *behaviour* is equivalent to there being no *prohibition* of that *behaviour*. Thus, a *rule* that applies to one *object*, that participation in a particular *type* of *interaction* is allowed, is not inconsistent with a *rule* that applies to another *object*, that an *interaction* of that *type* shall not occur. The latter rules out *interactions* of that *type* between those two *objects*, but the first *object* may participate in *interactions* of that *type* with some third *object*.

Example – The *enterprise specification* provides that *objects* in the *role*, orderPlacer, have *permission* to *establish communication* with *objects* outside the e.com *security domain*. An orderPlacer object attempts *communication* with an *object* representing a particular supplier system. securitySubsystem prevents that *communication* because of a temporary specific *prohibition* on *communication* with that supplierSystem object, which currently appears to be a zombie in a denial of service attack on e.com. This is not a *violation* of the *permission* of that orderPlacer object.

B.1.4.5 Prohibition [2-11.2.6]

A *prohibition* is a prescription that a particular *behaviour* shall not occur. A *prohibition* is equivalent to there being an *obligation* for the *behaviour* not to occur. Occurrence of that *behaviour* is a *violation*.

Example – The *enterprise specification* of the e-commerce system includes a *rule*, which prescribes that a salary record may be displayed only for a salary administrator, an auditor, or a manager of that employee. This *rule* is specified using the *roles*, salaryAdministrator, auditor, manager, and employee, the *record type*, salaryRecord, and the *action type*, recordDisplay.

Another *rule* prescribes that no *subsystem* may *establish communication* with a *system* outside the e-com *administrative domain* without a *permission* included in that specification or granted by securitySubsystem. Any such *communication* in the absence of a *permission* is a *violation*.

B.1.4.6 Violation [6.4.3 and 7.9.3]

A *violation* is an *behaviour* contrary to a *rule*. A *violation* of a *rule* that is part of a *contract* is a *failure*. [2-13.5.1]

Example – An *action* of securitySubsystem, which prevents an auditor object from examining a certain record, is a *violation* the *rule*, mentioned above, that a *party* in the role of auditor is *authorized* to examine any record in the e-commerce system. An *action* by a program executed by a databaseAdministrator object, which displays an employee salary, is a *violation* of the *rule*, mentioned above, that prohibits display of a salary record except for *objects* fulfilling certain *roles*.

An *enterprise specification* may include a *rule* prescribing certain *types* of *actions* to be taken by an *object* in the event of certain *types* of *violations*. That rule is an *obligation*, which applies to that *object*. Failure to take the prescribed *actions* is an inaction, and a *violation* of that *rule*.

Example – When an order is accepted before 4 PM, but ordered widgets, which are in stock and not already scheduled for shipment at the time of acceptance of that order, are not scheduled for shipment the same day, this behaviour by the shippingSubsystem is a *violation* of the *rule*, mentioned above, that obliges shippingSubsystem to schedule such shipments on the same day.

B.1.5 Accountability

B.1.5.1 Accountability [6.5 and 7.10]

An *action* of an *object* is a model of something that happens in the world of the *ODP system*. Things that happen include things done by people and organizations and by computer systems. They include things that happen with the participation of more than one person, organization, or computer system. In most cases people and organizations may be held accountable for what they do. The owner or operator of a computer system should be held accountable for what that system does. This document provides concepts for the specification of *accountability* for *actions*. [6.4]

To say an organisation does something is to use a convenient shorthand which means that some person or persons or some automated system does something, and that this is regarded for certain purposes as having been done by the organisation. *Accountability* concepts of the *enterprise language* allow for the *specification* to provide for the determination of the persons or organizations to be held accountable for certain *actions*. This is represented in that specification as *accountability* of the *objects* representing those persons or organizations.

Example — If a customer makes a purchase from an e-commerce system, this is represented in the *enterprise specification* as an *action* of the *object* representing that customer.

If the purchasing agent of e.com makes a purchase (in her capacity as purchasing agent), this may be represented as an *action* of the *object* representing e.com, an action of the *object* representing the person (the purchasing agent), or both. The choice depends on the purposes of the specifier or owner of the system.

If a person or organization causes a computer system to do something on its behalf, then the person or organization is accountable for what that system does. When the computer system does something, that has the same force as if the person or organization that caused the computer system to be in operation or allowed it to continue in operation had done that thing. For this reason, an *action* in an *enterprise specification* that represents something done by a computer system acting for a person may be an *action* for which that person is accountable. This is represented by a specification that the *object* representing that person is *accountable*.

Example — If a firm's computer system sends an order to the e-commerce system (and has been delegated authority to do so) this is represented as an *action* for which the *object* representing that firm is *accountable*.

This document provides rules for the specification of an *ODP system* in terms of the *accountability* for something which that *system* does. When the *ODP system* is acting on behalf of a person or organization, accountability for any thing the system does is determined by delegation by that person or organization to the system. This is represented in the specification by a *delegation* by an *object* representing that person or organization to the *object* representing that system.

B.1.5.2 Party [6.5.1 and 7.10.4]

An *ODP system* operates in a world that includes computer systems (that is, computers and the software they execute) and other automated systems. That world also includes natural persons, groups of natural persons and other entities considered to have some of the rights, powers and duties of a natural person (for example, corporations, governments, and other organizations). *Parties* are models of these entities. In some cases, these persons or other entities may cause that *ODP system* to do something on their behalf. The *enterprise language* provides concepts and structuring rules for specifying the effect in the world when this happens.

Examples — The *objects* representing people and the *objects* representing e.com and other firms are *parties.*

B.1.5.3 Commitment [6.5.2 and 7.10]

An *action* may result in an *obligation* by one or more of the participants in that *action* to comply with a *rule* or perform a *contract*. Such an *action* is a *commitment*. In an *enterprise specification* the *enterprise object(s)* participating in a *commitment* may be *parties* or *agents* acting on behalf of a *party* or *parties*. In the case of an *action* of *commitment* by an *agent*, the *principal* becomes obligated. (See B1.5.5 and B.1.5.6 below.)

Example — A firm may operate a computer system which sends an order to the e-commerce system The sending of the order by that computer system commits that firm to pay for the goods when timely delivered. (For example, e.com may have a contract with that firm which provides that, or this may the result of the operation of commercial law.) The sending of that order is represented as a *commitment.*

B.1.5.4 Declaration [6.5.3and 7.10]

Sometimes, when some person says something, the very fact of saying that causes a change in the world. The act of making such a statement may be represented as a *declaration*. The essence of a *declaration* is that, by virtue of the *action* of *declaration* itself and the authority of the *object* or its *principal*, it causes a state of affairs to come into existence outside the *object* making the *declaration*. An *ODP system* may be *delegated* by a *party* to participate in some *action* that is a *declaration.*

Example – e.com has agreements with its customers, which provide that, when an order is cancelled within twenty-four hours of the scheduled shipping data, e.com has the option to make a restocking charge of 5% of the invoice amount for the cancelled order. The e-commerce system is programmed to automatically make that charge in the case of a customer that has accounts receivable by e.com which are more than sixty days late in payment. The cancellation of an order by a firm, F, is represented in the specification as an *action*, cancel, by an *object*, firmF, referring to another *object*, order. An *action* of e-system invoking the restocking charge when an order is cancelled by firmF is a *declaration*. That *action* establishes a state of affairs in the *environment* of e-system: firm F is obliged to pay that charge to e.com.

B.1.5.5 Delegation and Authority [6.5.4 and 7.10.1, 7.10.2]

In an *enterprise specification* of an *ODP system*, that *system* is represented as an *object*. The specification describes the authority *delegated* to that *object*.

Example – The *specification* provides that e-system may be *delegated* the authority to cancel a contract with a supplier or customer of e.com (subject to the terms of that contract). Sending a message to the counterparty cancelling a contract is represented as a *declaration*: the *communication* of that message causes the contract to be cancelled. This will be the case, for example, when the e-commerce system automatically cancels an order.

Subclauses 7.10.1 and 7.10.2 prescribe rules for specification of *delegation* and authority

B.1.5.6 Agent and Principal [6.5.7, 6.5.8, and 7.10]

By each *delegation*, that system becomes an *agent* of the *party delegating*, and the *party* becomes a *principal* of the system.

Examples:

1 – In our *enterprise specification*, e.com is represented as an object, e.com. The person acting as chief financial officer (CFO) of e.com causes the e-commerce system to serve offered prices to the computer systems of customers (including web browsers and purchasing systems), which prices are determined by the CFO and entered into the e-commerce system.

This setting of offered prices is represented as an *action* of the *party*, CFO, representing the CFO. The CFO may delegate to the e-commerce system the authority to set prices during evenings and weekends. The CFO may also delegate authority to that system to further *delegate* this authority to an independent but federated pricing service in certain cases. e-system is the *agent* of CFO and CFO is the *principal* of e-system. If delegated by e-system pursuant to that authority, pricingServiceP (representing a federated pricing service) becomes the *agent* of CFO and the CFO is the *principal* of pricingServiceP.

In a different *enterprise specification*, it may instead be e.com that *delegates* to e-system the authority to set prices during evenings and weekends. e.com may also *delegate* authority to the e-system to further *delegate* this authority to pricingServiceP. e-system is the *agent* of e.com and e.com is the *principal* of e-system. If that authority is delegated by e-system to pricingServiceP, then pricingServiceP is the *agent* of e.com and e.com is the *principal* of pricingServiceP.

2- Another *ODP system* provides an e-commerce service, which an application service provider offers to many firms. Each firm using that system is represented in the specification of that system by an object in the role, firm.

The *enterprise specification* of that e-commerce service provides a means for firm to *delegate* the authority to set prices. If e.com uses that service, during the operation of that system, the object, e.com, fulfilling the role, firm, may *delegate* the authority to set prices to its pricingManager (a *role* for an *object* representing the person in the firm with authority to set prices). In the matter of setting prices, pricingManager is the *agent* of e.com.

Actions of the *object* representing an *ODP system* that are an exercise of *delegated* authority are *actions* that result in *accountability*. These *actions* represent something the *ODP system* does, which causes a state of affairs in that part of the universe of discourse represented by the *parties* in the *environment* of that *object*. The accountability concepts [6.5] are used to specify which kinds of states of affairs the thing represented by the *action* causes.

Example — If pricingService changes an offeringPrice, this *action* is a model of something that happens in the world of e-commerce. What happens is that there is a new state of affairs: e.com is now offering to sell at the changed price.

In a fully automatic e-commerce system, the *enterprise specification* will provide that the posting of a changed price object by an *agent* of e.com (for example, the pricingService) constitutes an offer by e.com to sell at the price indicated by that price. In that case, the effect of the *action* by the pricingService to change that price is this: e.com has made an offer and is committed to sell at the price indicated by price to any customer accepting the offer before it is withdrawn. (This is the representation in the specification of the situation in the world, and exactly represents that situation: The effect of the posting by the federated pricing service of the changed price is this: e.com has made an offer and is committed to sell at the price indicated by the posted price to any customer accepting the offer before it is withdrawn.)

This offer (the posting of the changed price object) is an *action* of an agent of e.com, for which e.com is accountable. In this case the *action* is a *commitment*, as e.com is obligated to sell at that price if the offer is accepted.

Since the e-commerce system is a fully automatic system operating over the internet, an acceptance of an offer to sell will be transmitted to the e-commerce system by software connected to the internet, perhaps a web browser. In an *enterprise specification* that includes customers and their systems, a webBrowser is the *agent* of an *object* in the *role*, customer and acts for the customer (the web browser acts for the customer by sending the order message when the customer clicks the Buy button).

The parts of an *ODP system* may be specified in this same way.

Example — Within the pricingService, the *object*, priceSelector, may *delegate* current market analysis to one or another marketAnalysisSubsystem depending on the *type* of *product*.

Some *actions* represent things that persons or organizations do, for which they are held accountable. *Delegation* has the effect of assigning responsibility to the *entity* represented by an *agent*, while not removing the accountability of the *entity* represented by the *principal*.

B.1.5.7 Evaluation [6.5.5 and 7.10]

An action that assesses the value of something is an evaluation. In an evaluation, the *ODP system* assigns a relative status to some thing, according to estimation by the system of usefulness, importance, preference, acceptability, etc.

Example — In order to replenish inventory, the e-commerce system prepares requests for bids and transmits them to widget suppliers. When bids are received, the e-com purchasing agent determines which bid or bids to accept. The decision considers estimation of the value of each bid, which the e-commerce system prepares. The e-commerce system assigns a relative status to each bid, according to some algorithm that produces an estimate of the value of that bid, using, not only the price, but the delivery terms offered, records of the previous on-time performance of that supplier, and records of receiving inspection reports on the quality of widgets from that supplier. This assignment of status is an *evaluation* by e-system of the bids.

Other examples of automated *evaluation* are credit scores and insurance underwriting ratings.

B.1.5. 8 Prescription [6.5.6 and 7.10]

Any *action* that sets a new *rule* or changes or removes an existing *rule* is a *prescription*. This includes a *prescription* that an existing *policy* is to be applied in a certain type of circumstance. *Prescriptions* may be made by a *party* with authority to do so, or by another *object*, as *agent* of that *party*.

Example — The CFO of e.com from time to time sets *rules* of the *policy* governing the granting of credit by the e-commerce system to established customers. The e-commerce system also includes a credit history evaluation *subsystem*, which is *delegated* authority by the CFO to change some rules of the credit policy. Such *actions* by CFO and creditHistoryEvaluationSubsystem are *prescriptions*.

B.1.6 More about community

B.1.6.1 Establishing a community [7.6.1]

A *community* may be established during the operation of an *ODP system*. The specification of that *system* will include *establishing behaviour*, which puts in place the *contract* of a *community*. [13.2.1]

Example – The specification of our e-commerceCommunity includes behaviour to establish a community of type, just-in-timeCommunity, with a supplier object representing a widget supplier who agrees to maintain inventory of widgets sold by e.com, and deliver them to e.com warehouses as directed by the inventoryMaintenance object. When a supplier agrees to do so, a new community is established, comprising the supplier object and the inventoryMaintenance object.

B.1.6.2 Assignment policy [7.6.2]

A *community* may include *rules* for choosing the *objects* that fulfil *roles* in that *community*. These *rules* may be called the assignment *policy*. An *assignment rule* prescribes, for some *role*, some characteristic that an *object* shall have to fulfil that *role*, or some characteristic that an *object* may not have if it is to fulfil that *role*. These *rules* specify the way in which objects are introduced and fulfil roles in that community.

Examples – The *assignment policy* of our e-commerceCommunity includes a *rule* providing that for an *object* to fulfil the customer *role* it shall be an *authenticated object*. (Our specification includes the specification of a securitySubsystem, which provides an *authentication function* for *objects* connecting to the e-commerce system.)

The specification includes *rules* for assigning *parties* to the *role*, widgetSupplier. When e.com agrees to purchase from a new supplier, a new *party object* is *created* in e-system to represent that supplier and that *party object* fulfils the *role*, widgetSupplier. The specification includes *rules* for *introducing* new *objects*. When e.com agrees to purchase from a new supplier, an object representing that suppliers system are introduced and fulfil the role supplierSystem.

The specification includes *rules* for assigning existing employee objects to the role, manager.

The assignment of a *role* in a *community* to a particular *object* may result in a conflict between the *policies* of that *community* applying to that *role* and the *policies* applying to some other *role* (in that *community* or some other *community*) which is already assigned to that *object*. An assignment *policy* may include rules for handling such conflicts. [7.9.1]

B.1.6.3 Relationship between communities [7.3.2, 7.8.3]

A *community* may *interact* with other *communities* in several ways:

1) An *object* representing a *community* (a *community object*) may fulfil a certain *role* in another *community*, *interacting* with other *objects* in that other *community*.

Example – e-system is a *configuration of objects*, including a purchasing *subsystem*, a shipping *subsystem*, and an administration *subsystem*. The *enterprise specification* includes a *community*, e-systemCommunity; the *objects* of that *community interact* to realize the *objective* of that *community*. e-system is a *composite object*, a *composition* of the *objects* of e-systemCommunity. The *enterprise specification* also includes supplyCommunity. The *objects* of that *community* are e-system, *objects* representing the sales systems of companies that supply e.com, and *objects* representing systems of other companies that are also customers of the supplying companies. When e-system participates in supplyCommunity, this is an *interaction* of supplyCommunity and e-systemCommunity.

2) Two or more *community objects* may *interact* in fulfilling *roles* in another *community*;

Example – The *enterprise specification* includes a *community* corresponding to each of the *subsystems* of e-system, each with its own *objective*, *policies*, etc. So there are purchasingCommunity, shippingCommunity, warehouseCommunity, and accountingCommunity. Each of these *subsystems* is also a *configuration of objects* (the parts of that *subsystem*); each of these *objects* fulfils one or more *roles* in the corresponding *community*. When *object* acting in a *role* in warehouseCommunity, the inventoryMaintenance object, interacts with an object fulfilling a *role* in the purchasingCommunity, the role, supplyPlanning, by providing information for use in purchasing widgets to restock inventory, this is an *interaction* between warehouseCommunity and purchasingCommunity.

3) The same *object* may be required to fulfil a *role* in two *communities*, thus introducing implicit *interaction* between those *communities* because of the shared *object*.

Example – When an *object* acts in the role, inventoryMaintenance, in warehouseCommunity and that object also fulfils the role, assetReporter, in accountingCommunity, by providing, for use in daily asset accounting, information that object obtains while fulfilling the role, inventoryMaintenance, this is an *interaction* between warehouseCommunity and accountingCommunity.

4) An *object* in a *community* may fulfil an *interface role*, through which the *community interacts* with *objects* in its *environment*, which *environment* includes an *object* fulfilling an *interface role* in another *communiy*;

Example – e-system may contain an *object* dedicated to monitor the business activities performed in e-commerceCommunity. This *object* may process the data it gathers and produce reports of preferred supplier ranking and active buyer ranking. There is also a ratingServiceCommunity. These two *communities* are operated independently, but exchange information. For that purpose, e-commerceCommunity specifies an *interface role*, fulfilled by the object, monitor, that provides local ranking information and receives nation-wide ranking information. The ratingServiceCommunity specifies an *interface role*, fulfilled by an *object* in that *community*, which receives company specific ranking data and provides nation-wide ranking information. These two *interface roles* enable *interaction* between the *communities*.

For each of above cases, when *object(s)* are involved in *interaction* between *communities*, an *enterprise specification* will include *policies* such as the following:

In case 1) the *community object* is subject to the *policies* of the *community* in which it fulfils a *role*.

Example – e-system (a *community object*) is subject to the *policies* of supplyCommunity.

In case 2) the community objects interacting by fulfilling roles in another community are subject to the policies of that community.

Example – In addition to being subject to their own individual policies, all the community objects representing the different subsystems of e-system, (purchasingCommunity, shippingCommunity, warehouseCommunity, etc.) are subject to the policies of e-system.

In case 3) the *object* that is fulfilling a *role* in both *communities* is subject to the *policies* of both *communities*. Other *objects* may be subject only to the *policies* of one of those *communities*.

Example – The *object* fulfiling both of the *roles*, inventoryMaintenance and assetReporting is subject to the *policies* of two different *communities* at the same time, warehouseCommunity and accountingCommunity. Other *objects* in warehouseCommunity may be subject only to the *policies* of warehouseCommunity .

In case 4) the *object(s)* fulfilling an *interface role* for each *community* may themselves form a *community* with a *policy* specified for their interactions. Those *objects* are each subject to the *policies* of the *community* in which they fulfil an *interface role* and to the policy specified for their interactions.

Example – The *object*, monitor, is subject to the *policies* of e-commerceCommunity and to the *policies* of the information exchange *community* composed of monitor and the *object* in the *interface role* of ratingServiceCommunity.

A fifth case of *community interaction* involves creation of a *community*. [7.3.2] Once a new *community* is created, the relationship between the newly created *community* and other *communities* will be one or more of the above, and these policy-community relationships will apply.

Example – In addition to a closed business-to-business supply chain community, the e-commerce system may make use of the web services infrastructure, by registering its services to an open registry, in order to find new potential customers. And, when the needs for customers are satisfied, e-commerce system may un-register the services from the open registry. Registration with the open registry is represented as *establishing behaviour*, and the open registry, the e-commerce system, and the potential customers using the registered web services are represented as a newly created *community*.

B.1.6.4 Domain [2-10.3]

A *domain* is a set of *objects* that are in some way controlled by another *object*, the *controlling object* of that *domain*. For each *domain* in an *enterprise specification*, that *specification* identifies the *controlling object* of that *domain*, the controlled *objects*, and the relationship between the *controlling object* and the controlled *objects*, which characterizes that domain.

Example – The parts of the e-commerce system are all under control of e.com. Other *domains* in that *system* include:

– a security *domain*, comprising *objects* providing data access and *communication* services, subject to *policies* set by a security authority *object*, securitySubsystem.

– a naming *domain*, with named *objects*, which obtain their names from a naming service *object*.

– an audited *domain*, comprised of *objects* which are audited by a certain auditor object.

B.1.6.5 Federation [3-5.1.2]

A *federation* is a *community* that includes *objects* that are in different *domains*. Because such *objects* are controlled in some way by different *objects*, there are concerns to be taken into account in forming a *federation* that are in addition to those that arise in forming a *community* of *objects* all in the same *domains*.

Example – The automated systems in our example are not all under common control. The e-commerce system is under control of e.com. It shall interact with, for example, another automated system controlled by a customer. We have many *administrative domains* in our specification; these include the *domain* of automated systems controlled by e.com, as well as a domain controlled by each customer. The e-commerce system also interacts with people; each person is, ultimately, controlled only by herself or himself. (Although, we might, if it is useful in our specification, consider *objects* representing employees to be members of *domains* controlled by *objects* representing their employers.)

The *objective* of a *federation* is often to join two or more *<X>-domains* into a larger, integrated *domain*, that is, an <X>-*federation*. The notion of *<X>-domain* expresses a *community* managed by a single authority with respect to the characterizing relationship, X. [2-10.3]

Example – The *objects* of e-commerceCommunity are not all under common control. The supplier systems are under administrative control of each of the suppliers. There are several immediately visible "domains", for example, the e.com domain(s?), and the customer IT system domain. Within the e-system itself, there are also domains under separate control. All objects in the e-system are in a single securityDomain, with *characterizing relationship*, subjectToSecurityPolicySetBy and *controlling object*, securitySubsystem. But the objects of purchasingSubsystem and shippingSubsystem are in a policy *domain*, with *characterizing relationship*, setsPoliciesFor, with *controlling object*, fulfillmentDivisionExecutive, while the objects of securitySubsystem are in a different policy domain, with *controlling object*, CIO.

An *enterprise viewpoint* specification might specify a number of separate federations, such as document management federations, billing management federations, customer management federations and so on. Or it might take a unified approach, in which all these aspects are captured into a single federation community. This federation community might also include a mechanism for incorporating additional functionality, to achieve a new, shared objective.

B.2 Second Example – Specification of a Library

This section provides a second example to illustrate the use of the *enterprise language* concepts and structuring rules to specify an *ODP system*. The example describes the elements that comprise an enterprise specification of a Library. Thus, in this case, the *ODP system* is a business system (which may or may not include a computer system). The example shows how RM-ODP specifications (in particular, those from the *enterprise viewpoint*) can be used to specify systems other than computer systems.

Example – This example is based on a university library, especially on the regulations that rule the process of borrowing items from that library. Instead of a general and abstract *system*, this example is loosely based on the regulations defined for the Templeman Library at the University of Kent at Canterbury, a *system* that has been previously used by different authors for illustrating some of the ODP concepts. The rules that govern the borrowing process of that library *system* are as follows:

1. Borrowing rights are given to all academic staff, and to postgraduate and undergraduate students of the University.
2. There are prescribed periods of loan and limits on the number of items allowed on loan to a borrower at any one time. These 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.
3. Items borrowed shall be returned by the due date and time.
4. 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.
5. Failure to pay charges may result in suspension by the Librarian of borrowing facilities.

NOTE - Unless otherwise stated, all references in this section to the Library community will refer to the *community* representing the Templeman Library and its environment, whose borrowing regulations have been described above. Other *communities* will also be mentioned in this Annex, such as the University community that represents the University that the library serves, and a banking community representing the use by the library of services offered by banks. Some variations of the Templeman Library *community* will also be introduced for illustration purposes. For instance, for illustrating the *accountability concepts* of the *enterprise viewpoint language*, we will introduce another library that uses a computerized system to keep track of the borrowers and their outstanding loans.

B.2.1 Enterprise Specification

The *enterprise viewpoint* focuses on the *purpose* (i.e., *objective*), *scope* and *policies* for the *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, or the technology used to implement it.

Four key concepts of *enterprise language* are: *system*, *scope*, *enterprise specification*, and *field of application*. The following points in this section (B.2.1) describe these four concepts. Sections B.2.2 to B.2.4 focus on the elements that constitute the *enterprise specification* of the *system*: *communities*, *behaviour*, and *policies*. Section B.2.5 concentrates on the *accountability* concepts. Finally, Section B.2.6 discusses further issues related to *communities*, such as their lifecycle and their interactions.

B.2.1.1 System

Example – In our example, the *system* we want to specify is a university library, in particular (a reduced and simplified version of) the Templeman Library at the University of Kent at Canterbury, whose borrowing regulations have been given above. This *system* (hereinafter called the 'Library System', or 'LS') and its *scope* are described by an *enterprise specification.*

B.2.1.2 Scope [6.1.1]

The *scope* of a *system* is defined in terms of its intended *behaviour*; which is expressed in the *enterprise language* in terms of *roles* (see B.2.3.2) or *processes* (B.2.3.5) or both, *policies* (B.2.4.2), and the relationships of these. All these elements will be described below.

Example – The *scope* of the LS describes its expected *behaviour*, i.e., the way it is supposed to work.

B.2.1.3 Enterprise specification [3-4.2.2]

An *enterprise specification* of an *ODP system* describes it from the *enterprise viewpoint*, and is composed of the specifications of the following elements: *communities* (B.2.2.1), *roles* (B.2.3.2), *processes* (B.2.3.5), *policies* (B.2.4.2), and their relationships.

Example – The LS and the environment in which it operates are represented as one *community*, the libraryCommunity. The *enterprise specification* will state the *objective* of that *community*, how it is structured, what it does, and what *objects* comprise it.

B.2.1.4 Field of application

The *field of application* of an *enterprise specification* describes the properties that the *environment* of the *ODP system* shall have for the specification to be used.

Example – The library in the example is the library of a University, and therefore assumes the existence of some structures and roles typical of those organizations. It may not make sense if we try to use this specification for a library in the Army, where no such concepts as academic staff or students can be easily applied.

B.2.2 Community

B.2.2.1 Community [3-5.1.1]

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* (B.2.2.3) governing their collective *behaviour* (B.2.3.1), and that has been formed for a particular *objective* (B.2.2.2).

Example – In the specification of the Library System (LS), the library is represented as a *community*; the behaviour of that community is specified in the following using *roles*, *processes*, and *polices.*.

B.2.2.2 Objective [6.2.1]

An *enterprise specification* states the *objective* of each of the *communities* that comprise it.

Example – The LS maintains a collection of books, periodicals, and other items, that may be borrowed by its members. The LS comes into being with the establishment of its collection, with the primary *objective* of "sharing this collection amongst the University members".

B.2.2.3 Contract [2-11.2.1]

The *contract* of a *community* specifies the *objective* of that *community* and how this *objective* can be met. That *contract* is specifies the different *roles* that *objects* may fulfil in the *community* (i.e., its structure and behaviour) and the *policies* that govern the *behaviour* of these *objects* while fulfilling *roles* in the *community*.

Please notice the importance of this concept within a *community* specification, since the *contract* contains all the information about the structure of a *community*, its *behaviour*, and the way it operates.

B.2.3 Behaviour

B.2.3.1 Behaviour

In the *enterprise language*, behaviour is specified using *roles*, *processes*, or both, *policies*, and the relationships among these.

Roles identify abstractions of the *community* behaviour, and are fulfilled by *objects* in the *community*. *Processes* describe the *community behaviour* by means of (partially ordered) sets of *steps*, which are related to achieving some particular sub-*objective* within the *community*. A *step* is an *abstraction* of an *action*, which hides (some of the) *objects* participating in that *action*.

B.2.3.2 Role [2-9.14]

Example – From the regulations of the Templeman Library, three main *roles* can be identified in our LS, namely borrower, library item, and librarian. There are three special kinds of borrowers (academic, undergrad, and postgrad), and two kinds of items (book and periodical). Loans are *enterprise objects* that are *artefacts* in borrow and return *interactions* between borrowers and a librarian. Another *role*, calendar, is fulfilled by an *enterprise object* that deals with the passage of time (e.g. a wall clock).

NOTE – In this Annex B.2, the use of the name of a *role* is a reference to an *enterprise object* fulfilling that *role*.[see also 3-5.2 Note 3]

B.2.3.3 Enterprise object

ODP systems are modelled in terms of *objects*. An *object* is a model of an *entity*. *Enterprise objects* model the *entities* defined in an *enterprise specification*.

Example – The libraryCommunity is composed of *objects* fulfilling the *roles* identified above. In this specification *objects* represent people (teachers, students, persons working as librarians…), books, periodicals, loans, clocks, etc. Please notice that an *object* may fulfil more than one *role*, as it happen for instance when a person can be both a librarian and a student of the University (if the University regulations allow this to happen).

The library can also be represented as an *object*; for example, it may be abstracted as a *community object* that may fulfil a *role* in another *community* (see B.2.6.3).

B.2.3.4 Action

Example – The following *actions* can be identified from the regulations in the example:

(a) A borrower borrows an item with *permission* of the librarian,

(b) A borrower returns a borrowed item to the librarian,

(c) A librarian fines a borrower,

(d) A fined borrower pays his/her fines to the librarian, and

(e) The librarian suspends a borrower for being late in paying his/her fines.

B.2.3.5 Process and step [6.3.5]

Processes may also be defined in an *enterprise specification* for describing *behaviour*.

A *process* is "a collection of *steps* taking place in a prescribed manner and leading to an *objective*" [6.3.5].

Example – In the Library System, the *process* that defines the normal way of operation of users has two *steps*: (1) a borrower borrows an item, and (2) a borrower returns the borrowed item before its due date.

This *process* specifies the order in which the *steps* shall occur, and leads to the *objective*: of the LS: "to share the collection of items amongst the University members".

Please notice how each *step* in this *process* is an *abstraction* of an *action*, where some of the participants in that *action* can be left unspecified (e.g. the librarian).

B.2.3.6 Enterprise object and action

For every *action*, there is at least one participating *object* fulfilling at least one *role* in some *community*.

The involvement of an *object* depends on the kind of *role*.

Example – In the libraryCommunity, borrowers and librarians are *actors* in all *actions* specified for that *community*. Items are *resources*. Calendar is an *artefact* in the *action* of fining a borrower (c), since it is only referenced, but does not participate in that *action*.

B.2.4 Policy [2-11.2.7 and 6.4]

A policy is a set of *rules* that constrains the *behaviour* and membership of *communities* in order to make them achieve their *objectives*. The rules can be expressed as *obligations, authorizations, permissions,* or *prohibitions*. *Behaviour* contrary to a *rule* is a *violation*.

Example – The basic *policies* that govern the *behaviour* of the libraryCommunity were defined by the regulations at the beginning of this example.

Examples of membership *policies* are those that prescribe: each item can be borrowed by at most one borrower at a time; at least one *object* fulfils librarian in the library. (These two *policies* are *obligations*.) Other membership *rules* (although they may not apply in this example) might establish, for example, that a suspended borrower cannot work as a librarian (a *prohibition*).

Examples of enterprise *policies* that govern the *behaviour* of the *system* can be extracted from the Library regulations:

(1) Any borrower is *permitted* to borrow an item if the number of his/her borrowed items is less than his/her allowance (as provided in the regulations: eight items for undergraduates, etc.).

(2) An undergraduate is *prohibited* from borrowing a periodical item.

(3) Any borrower is *permitted* to borrow an item for a given period of time. The period depends on the kind of borrower and the kind of the item being borrowed.

(4) Any borrower is *obliged* to return his/her borrowed items before their due date.

(5) A librarian is *authorized* to fine a borrower who *violates* the previous *rule*, i.e., who does not return an item before its due date.

(6) Any fined borrower is *obliged* to pay his/her fines.

(7) A *violation* of the previous *rule* may result in an *action*, suspension of the borrower by the librarian, that is, the librarian is *authorized* to suspend a borrower who does not pay his/her fines.

Note that in these *policies* there are some details left unspecified, for example, when the *actions* after a *policy violation* are executed (expressed by “may result in” in policy 7), or the precise amount of the fines. Policy 7 also *authorizes* a librarian to suspend a late-payer, but it does not specify when; the librarian may not to suspend the borrower at all (the suspension action can be delayed forever). It may be a matter for further refinements to this specification to clarify all these open details.

Note also that most *policies* say what needs to happen when an *action* occurs, but not the reasons or circumstances that triggered that *action* in the first place.

It is important to notice that in most *systems* there are some *rules* that govern the *behaviour* of that *system* and which are not made explicit anywhere; yet they need to be explicitly stated in the *enterprise specification*.

Example – Such *rules* include the common sense *rules*: for instance, a borrower cannot return an item she has not previously borrowed.

Normally, the act of writing an *enterprise specification* helps uncovering many of these implicit *rules*. However, special care should be taken with these *rules*, since it is easy to under-specify, over-specify, or make wrong assumptions about them (“common sense is quite uncommon”).

B.2.5 Accountability [6.5 and 7.10]

An *enterprise specification* identifies 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. *Parties* have intentions, and are accountable for their *actions* (or failures to act).

Example – A borrower is *accountable* for the return of a book he has previously borrowed. He is not *accountable*, however, for the return of a book he has not borrowed. Likewise, the librarian is *accountable* for suspending those fined borrowers that do not pay their fines.

Authority or functions can be *delegated*. *Principal parties* are responsible for the *actions* of any *parties* acting as their delegated *agents*, including their possible *commitments*, *prescriptions*, *evaluations*, *declarations*, and further *delegations*.

Example – A borrower may instruct a friend to return one of his borrowed books if he is unable to return it before its due date (e.g. if he is going to be out of town when the loan expires). In this case, the borrower is still *accountable* for his original *commitment* to return the book, independently from the fact that he has *delegated* his *commitment* to his friend. If the friend forgets and fails to return the book on time, the borrower will still be liable for a charge for returning the book late.

B.2.5.1 Party [6.5.1 and 7.10]

Example – The *objects* representing people and the *objects* representing organizations (for example, the Library) are *parties*. The *objects* representing computer systems or machines (e.g. the clock object fulfilling calendar) are not *parties*.

An *enterprise object* that is not a *party* cannot be *accountable* for its *actions*; it always operates on behalf of a *party*.

Example – Suppose another library Z has a Library Support System (LSS), that is, a computer system (not modelled in this specification) in charge of keeping track of the collections of items owned by the Library Z, the outstanding loans, and the borrowers. Suppose that the LSS has an internet front-end for consulting whether a book is free or on loan, the status of a given borrower, etc. Then, the LSS (which is not a *party*) acts on behalf of its librarian (a *party*) when automatically answering the queries via its web interface. In this scenario, the librarian of Library Z is *accountable* for all the information provided by the LSS.

Certain kinds of *action* involve *accountability*. These include *commitment*, *declaration*, *delegation* and *prescription*.

B.2.5.2 Commitment [6.5.2 and 7.10.3]

Commitments are *actions* resulting in an *obligation* by one or more participants in the *action* to comply with a *rule* or perform a *contract*.

Example – The *action* of borrowing a book is an example of *commitment*, since the borrower *commits* to return the book before its due date.

B.2.5.4 Declaration [6.5.3 and 7.10.4]

Example – The librarian has the authority to suspend persons who do not pay their fines. An *action* declaring that a borrower is suspended is a *declaration*, and it calls for a set of *actions*, such as including him in the list of suspended persons, writing a letter notifying this fact, sending the letter to the borrower, etc.

B.2.5.5 Delegation and authority [6.5.4 and 7.10.1]

Example – A borrower may *delegate* another person to return a borrowed book.

As an example of *delegating* authority, suppose that the librarian of Library Z could *delegate* to the previously mentioned Library Support System (LSS) the authority to suspend borrowers who do not pay their fines. Thus, that librarian could, for example, instruct the LSS to automatically suspend borrowers with pending fines greater than 20 pounds, if their debt is older than one month. Once the Library Support System detects a borrower in these circumstances, it will automatically *declare* that borrower suspended.

Of course, once the user is declared suspended, the LSS will accomplish all the *actions* that the *declaration* calls for; but please notice that these subsequent *actions* are the result of the *delegation*. It is also worth pointing out that in this case, the *party* responsible for all the *actions* carried out by the LSS is the librarian.

B.2.5.6 Agent and principal

Example – In case of a borrower delegating another person to return a book before its due date, that person becomes an *agent* of the borrower, who is the *party delegating*, and the borrower becomes a *principal* of that person. The borrower remains responsible for the return of that book.

In case of a librarian object *delegating* authority to suspend users to the Library Support System of Library Z, the Library Support System, acting on behalf of the librarian ,will make the *declaration*. The System thus becomes an *agent* of the librarian, who is the *party delegating*, and the *party* becomes a *principal* of the Library Support System.

B.2.5.7 Prescription [6.5.6 and 7.10.5]

Prescriptions are *actions* that establish *rules*. It is important to note that *prescriptions* happen only if the *enterprise specification* provides for making *prescriptions*.

Example – In the regulations of the Templeman Library example nothing is mentioned about the possibility of establishing or changing *rules*, and therefore they *prescriptions* forbidden.

In another LibraryCommunity (say W), the regulations may establish that the librarian of Library W may change the period of loans during the Summer time, or the number of items that a borrower is allowed to simultaneously have on loan. The *action* of changing the loan periods and item limits is a *prescription*

Other regulations of Library W may also allow to add new *rules* to those governing the borrowing of items, if they: (a) do not conflict the existing rules; (b) are proposed by a Library member; and (c) are adopted at the Library's annual meeting by the majority of the Library members. The *action* of adopting a new *rule* is a *prescription*.

B.2.6 More About community [7.3.2]

B.2.6.1 Lifecycle of a community [7.6]

A *community* is created by *instantiating* the corresponding *contract template*. Instantiation of a *contract template* involves the assignment of *objects* to *roles*.

Example – The libraryCommunity comes into being with the establishment of its collection of items.

A *community* may include *behaviour* for creating new *communities*. Establishment of a *federation* creates a new *community*; this involves determining the *configuration* of that *community* and the *contract* of that *community*, including *policies* for that *community*.

Example – This is the case if a *federation* of university libraries is established to allow a wider exchange of books. In this new *community* (the *federation*), libraries from different universities can share books or borrowers, and books can be returned in libraries different to the one they are on loan.

B.2.6.2 Assignment rules [7.6.2]

A *community* may include *rules* for choosing the *objects* that fulfil *roles* in that *community*. These *rules* may be called the *assignment rules*. An *assignment rule* prescribes, for some *role*, some characteristics that an *object* shall have if it is to fulfil that *role*, or some characteristics that an *object* may not have if it is to fulfil that *role*.

Example – An *assignment policy* may state that for an *object* to fulfil librarian the person that *object* represents must have a contract with the University that allows that person to work in a position of that category and, in case he can be a valid borrower of that *community*, that he is not currently suspended. Likewise, for an *object* to fulfil borrower, the person represented by that *object* must be in possession of valid credentials as academic staff or student.

In general, an *object* may fulfil many *roles*, in any number of *communities*.

Example – A member of the university staff may fulfil a teacher *role* in an educational *community* and also fulfil borrower in the libraryCommunity. Alternatively, a single *role* may be fulfilled by more than one *object*: a library has many borrowers.

B.2.6.3 Relationship between communities

A complete *enterprise specification* may consist of a number of related *community* specifications. *Communities* may interact in several ways.

Case 1 – Interaction between *communities* happens when a *community object* fulfils one or more *roles* in another *community*.

Example – The *community object* composed of the libraryCommunity fulfils a borrower *role* in another library *community*. This may be the case when there are agreements between libraries of different universities for inter-library loan of books. Thus, if a borrower of libraryCommunity wants a book that is not currently available at the library, the specification may provide that the libraryCommunity *community object* may borrow the book from another library where the book is available.

Case 2 – Two *community objects* interact in fulfilling *roles* in another *community*.

Example – Suppose that libraries can buy and sell books through some book broker system. In this case, if two library *communities* are involved in a transaction, they both fulfil *roles* (buyer and seller) in a book-brokering *community*.

Case 3 – An *object* fulfils *roles* in two *communities*.

Example – So far, we have considered the library in isolation, but it is, of course, closely related to the University that it serves. In a University *community* we find *roles* such as student, researcher, teacher, headOfDepartment, supportStaffMember, etc. These *roles* are fulfilled by *objects* representing people, which *objects* may at some point also fulfil the *role* of borrower in the libraryCommunity. Thus, the same *enterprise object* (in this case, representing a person) may fulfil *roles* in different *communities*. In order to fulfil a borrower *role*, an *enterprise object* must fulfil an appropriate *role* in the university.

Case 4 – An *object*, in fulfilling a *role* of one *community*, interacts with an *object* fulfilling a *role* in another *community*. This kind of *interaction* is achieved using *interface roles* [B.2.6.4]. Please note that this kind of *interaction* can be within a larger *community*, or not (as it may be in the case in a business-to-business interaction).

Example – This is the case of a librarian *object*, which periodically may need to call the Support Centre of the University to set the time of the clock, hence interacting with *objects* fulfilling *roles* in *communities* external to the libraryCommunity.

Case 5 – A *community* may include *behaviour* for creating new *communities*. *Federation* establishment is an example of this situation, since it means the creation of a new *community* involving the definition of appropriate *policies*, the structure for that *community*, and the *community contract*.

Example – The creation of a *community* may happen when a library, X, has a specified *behaviour* for temporarily establishing a trading *community* with other libraries for finding a requested book, if it is not currently available at X. This newly created *community* would cease to exist as soon as one copy of book is located at another library, Y, and is borrowed by the original requestor (either directly from library Y, which owns the copy of the book, or through library X acting as a borrower of Y — see interaction case 1).

In this fifth case, once the new *community* is created, during its lifetime the relationship between the created *community* and others will fall under one of the four cases above.

In all kinds of interactions of *communities* it is critical to consider the *invariants* that determine the constraints on the collective *behaviour* of the *communities* concerned, and the *objectives* and *policies* that govern the different *communities*. The *communities* involved in an *interaction* may have differing *rules*; all of the *objects* participating in that *interaction* must be able to conform to all those *rules*.

For instance, where a *community object* fulfils one or more *roles* in other *community* (case 1), the *community* that *community object* represents is governed by the *policies* of the other *community*.

Example – This happens when a library acts as a borrower in another library, and then lends the book to a borrower of its own *community*. The *policies* that apply to this loan include the *policies* of both libraries.

In the case of an interaction between *communities* where the same *object* is required to fill specific *roles* in more than one *community* (case 3), an *invariant* specifies how the *actions* of that *object* affect those *communities*.

Example – Since the Templeman Library interacts with the University *community*, some of the University *rules* apply, such as etiquette and manners for dressing and behaving in public, holiday periods, etc. Likewise, the laws and customs of the country where the Library is based apply as well, forcing borrowers to pay fines in sterling pounds, speak in English to a librarian, etc.

NOTE – In this case the *composition* of *communities* introduces the *composition* of *policies*. If we consider a university library in an Polish-speaking country, the *assignment rules* for librarian will probably indicate that the *object* fulfilling that *role* must, in addition to complying with all the university and library regulations, be fluent in Polish.

Case 4 happens when an *object*, in fulfilling a *role* of one *community*, interacts with an *object* fulfilling a *role* in another *community*. This *interaction* can be within a larger *community*, or not. If the *interaction* is within a larger *community*, C, the *community objects* are subject to the *policies* defined for each *community*, and also to the *policies* defined for *community* C, in a way similar to case 2. If there is no such larger *community*, the *interactions* are subject to the individual *policies* defined for all the *interface roles* involved in the *interaction*

When composing *communities*, there will be a set of *policies* common to those *communities*. These *policies* shall be consistent, although unspecified *behaviour* in the composite *community* may allow room for (mutually inconsistent) *behaviour* in each individual *community*.

Example – One library specialized in particular topics may not allow borrowers to be under 18, while other library whose items are toys and books for children does not allow their borrowers to be over 12. There is no problem when federating these two libraries if there is no regulation in the *federation* about minimum age of borrowers, hence allowing the *communities* to co-exist without conflicts.

B.2.6.4 Interface role

The *enterprise language* introduces the concept of *interface role*, a *role* in a *community* identifying *behaviour* that takes place with the participation of *objects* that are not members of that *community*. *Interface roles* are used in case 4 of interaction between communities [B.2.6.3] whereby an *object*, in fulfilling a *role* in one *community*, interacts with *objects* fulfilling *roles* in other *communities*.

Example – As mentioned [B.2.6.3], this is the case for a librarian, who periodically may need to call the Support Centre of the University to set the time of the clock, hence interacting with *objects* fulfilling *roles* in *communities* external to the libraryCommunity.

INDEX