ISO TC 184/SC 5 N 522

Date: 2014-04-29

ISO/PDPAS 19450

ISO TC 184/SC 5/WG 1 N 522

Secretariat: ANSI

Automation systems and integration — Object-Process Methodology

Systèmes d'automatisation et intégration -- Méthodologie du processus-objet

Document type: Publicly Available Specification
Document subtype:
Document stage: (20) Preparatory
Document language: E
STD Version 2.1c2

Violators may be prosecuted.

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

In other circumstances, particularly when there is an urgent market requirement for such documents, a technical committee may decide to publish other types of normative document:

— an ISO Publicly Available Specification (ISO/PAS) represents an agreement between technical experts in an ISO working group and is accepted for publication if it is approved by more than 50 % of the members of the parent committee casting a vote;

— an ISO Technical Specification (ISO/TS) represents an agreement between the members of a technical committee and is accepted for publication if it is approved by 2/3 of the members of the committee casting a vote.

An ISO/PAS or ISO/TS is reviewed after three years in order to decide whether it will be confirmed for a further three years, revised to become an International Standard, or withdrawn. If the ISO/PAS or ISO/TS is confirmed, it is reviewed again after a further three years, at which time it must either be transformed into an International Standard or be withdrawn.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO/PAS 19450 was prepared by Technical Committee ISO/TC 184, *Automation systems and integration*, Subcommittee SC 5, *Interoperbility, integration, and architectures for enterprise systems and automation applications*.

This second/third/... edition cancels and replaces the first/second/... edition (), [clause(s) / subclause(s) / table(s) / figure(s) / annex(es)] of which [has / have] been technically revised.

Introduction

Object-Process Methodology (OPM) is a compact conceptual approach, language, and methodology for modelling and knowledge representation of automation systems. The application of OPM ranges from simple assemblies of elemental components to complex, multidisciplinary, dynamic systems. OPM is suitable for implementation and support by tools using information and computer technology. This document specifies both the language and methodology aspects of OPM in order to establish a common basis for system architects, designers, and OPM-compliant tool developers to model all kinds of systems.

OPM provides two semantically equivalent modalities of representation for the same model: graphical and textual. A set of hierarchically structured, interrelated Object-Process-Diagrams (OPDs) constitutes the graphical model, and a set of automatically generated sentences in a subset of the English language constitutes the textual model expressed in the Object-Process Language (OPL). In a graphical-visual model, each OPD consists of OPM elements, depicted as graphic symbols, sometimes with label annotation. The OPD syntax specifies the consistent and correct ways to manage the arrangement of those graphically elements. Using OPL, OPM generates the corresponding textual model for each OPD in a manner that retains the constraints of the graphical model. Since OPL's syntax and semantics are a subset of English natural language, domain experts easily understand the textual model.

OPM notation supports the conceptual modelling of systems with formal syntax and semantics. This formality serves as the basis for model-based systems engineering in general, including systems architecting, engineering, development, life cycle support, communication, and evolution. Furthermore, the domain-independent nature of OPM opens system modelling to the entire scientific, commercial and industrial community for developing, investigating and analysing manufacturing and other industrial and business systems inside their specific application domains; thereby enabling companies to merge and provide for interoperability of different skills and competencies into a common intuitive yet formal framework.

OPM facilitates a common view of the system under construction, test, integration, and daily maintenance, providing for working in a multidisciplinary environment. Moreover, using OPM, companies can improve their overall, big-picture view of the system's functionality, flexibility in assignment of personnel to tasks, and managing exceptions and error recovery. System specification is extensible for any necessary detail, encompassing the functional, structural and behavioural aspects of a system.

One particular application of OPM is in the drafting and authoring of technical standards. OPM helps sketch the implementation of a standard and identify weaknesses in the standard to reduce, thereby significantly improving the quality of successive drafts. With OPM, even as the model-based text of a system expands to include more details, the underlying model keeps maintaining its high degree of formality and consistency.

This Publicly Available Specification (PAS) provides a baseline for system architects and designers, who can use it to model systems concisely and effectively. OPM tool vendors can utilise the PAS as a formal standard specification for creating software tools to enhance conceptual modelling.

This Publically Available Specification provides a presentation of the normative text that follows the eBNF specification of the language syntax. All elements are presented in Clause 6 through 13 with only minimal reference to methodological aspects, Clause 14 presents the context management mechanisms related to in-zooming and unfolding.

This specification utilizes several conventions for the presentation of OPM. Specifically, Arial bold font in text and Arial bold italic font in figure captions, table captions and headings distinguish label names for OPM objects, processes, states, and link tags. OPL reserved words are in Arial regular font with commas and periods in Arial bold font. Most figures contain both a graphic image, the OPD portion, and a textual equivalent, the OPL portion. Because this is a language specification, the precise use of term definitions is essential and several terms in common use have particular meaning when using OPM. Annex B.6 explains other conventions for the use of OPM.

Annex A presents the formal syntax for OPL, in EBNF form.

Annex B presents conventions and patterns commonly used in OPM applications.

Annex C presents aspects of OPM as OPM models.

Annex D summarizes the dynamic and simulation capabilities of OPM.

Annex E presents a summary of the graph grammar of the OPD's.

The International Organization for Standardization (ISO) [and/or] International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this document may involve the use of a patent concerning OPM as a "Modeling System" given in Clause 6 through 14.

ISO takes no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured the ISO that he/she is willing to negotiate licences either free of charge or under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO . Information may be obtained from:

Prof. Dov Dori
Technion Israel Institute of Technology
Technion City
Haifa 32000, Israel
dori@ie.technion.ac.il

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. ISO shall not be held responsible for identifying any or all such patent rights.

ISO (www.iso.org/patents) and IEC (http://patents.iec.ch) maintain on-line databases of patents relevant to their standards. Users are encouraged to consult the databases for the most up to date information concerning patents.

Automation systems and integration — Object-Process Methodology

1 Scope

This Publicly Available Specification (PAS) specifies Object-Process Methodology (OPM) with detail sufficient for enabling practitioners to utilise the concepts, semantics, and syntax of OPM as a modelling paradigm and language for producing conceptual models at various extents of detail, and for enabling tool vendors to provide application modelling products to aid those practitioners.

While this PAS presents some examples for the use of OPM to improve clarity, this International Standard does not attempt to provide a complete reference for all the possible applications of OPM.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 14977, *Information technology — Syntactic metalanguage — Extended BNF*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply:

NOTE 1 To facilitate look up, terms are in alphabetical sequence.

NOTE 2 *Italicized* words in the definitions are themselves terms defined in this clause.

3.1
abstraction, noun
outcome of an *abstraction process*

3.2
abstraction, verb
decreasing the extent of detail and *system* model *completeness* in order to achieve better comprehension

3.3
affectee
transformee that is affected by a *process* occurrence, i.e. its *state* changes

NOTE An affectee can only be a stateful object. A stateless object can only be created or consumed, but not affected.

3.4
agent
enabler that is a human or a group of humans

3.5
attribute
object that characterizes a *thing* other than itself

3.6
behaviour
transformation of *objects* resulting from the execution of an *OPM* model comprising a collection of *processes* and *links* to *objects* in the model

3.7
beneficiary
<system> stakeholder who gains *functional value* from the *system's* operation

3.8
completeness
<system model> extent to which all the details of a *system* are specified in a model

3.9
condition link
procedural link from an *object* or object *state* to a *process*, denoting a procedural constraint

3.10
consumee
transformee that a *process* occurrence consumes or eliminates

3.11
context
<model> portion of an *OPM* model represented by an Object-Process Diagram and corresponding Object-Process Language text

3.12
control link
procedural link with additional control semantics

3.13
control modifier
symbol embellishing a link to add control semantics to it, making it a control link

NOTE The control modifiers are the symbols 'e' for *event* and 'c' for *condition*

3.14
discriminating attribute
attribute whose different *values* identify corresponding specialization relations

3.15
effect
change in the *state* of an *object* or an *attribute value*

NOTE An effect only applies to a stateful object.

3.16
element
thing or *link*

3.17
enabler
<process> *object* that enables a *process* but which the *process* does not *transform*

3.18
event
<OPM> point in time of creation (or appearance) of an *object*, or entrance of an *object* to a particular *state*, either of which may initiate an evaluation of the *process precondition*

3.19
event link
control link denoting an *event* originating from an *object* or object *state* to a *process*

3.20
exhibitor
thing that exhibits (is characterized by) a *feature* by means of the exhibition-characterization relation

3.21
feature
attribute or *operation*

3.22
folding
mechanism of *abstraction* achieved by hiding the *refineables* of an *unfolded refinee*

NOTE 1 The four kinds of folded refineables are parts (part folding), features (feature folding), specializations (specialization folding), and instances (instance folding).

NOTE 2 Folding is primarily applied to objects. When applied to a process, its subprocesses are unordered, which is adequate for modelling asynchronous systems, in which processes' temporal order is undefined.

NOTE 3 The opposite of folding is unfolding.

3.23
function
process that provides *functional value* to a *beneficiary*

3.24
general, noun
<OPM> *refineable* with specializations

3.25
informatical
of, or pertaining to informatics, e.g., data, information, knowledge

3.26
inheritance
assignment of OPM *elements* of a *general* to its specializations

3.27
input link
link from *object* source (input) state to the transforming *process*

3.28
instance

<model> *object instance* or *process instance* that is a *refinee* in a classification-instantiation relation

3.29
instance
<*operational*> *object instance* or *process instance* that is an actual, uniquely identifiable thing that exists during model operation, e.g., during simulation or runtime implementation

NOTE A process instance is identifiable by the operational instances of the *involved object set* during process occurence and the process start and end time stamps of the occurrence.

3.30
instrument
non-human *enabler*

3.31
invocation
<process> initiating of a *process* by a *process*

3.32
involved object set
union of *preprocess object set* and *postprocess object set*

3.33
in-zoom context
things and links within the boundary of the *thing* being *in-zoomed*

3.34
in-zooming
<object> *object* part *unfolding* that indicates spatial ordering of the constituent *objects*

3.35
in-zooming
<process> *process* part *unfolding* that indicates temporal partial ordering of the constituent *processes*

3.36
link
graphical expression of a *structural relation* or a *procedural relation* between two *OPM things*.

3.37
metamodel
model of a modelling language or part of a modelling language

3.38
model fact
relation between two *OPM things* or *states* in the *OPM* model

3.39
object
<OPM> model element representing a thing that does or might exist physically or *informatically*

3.40
object class
pattern for *objects* that have the same *structure* and pattern of *transformation*

3.41
Object-Process Diagram
OPD
OPM graphic representation of an *OPM* model or part of a model, in which *objects* and *processes* in the universe of interest appear together with the *structural* and *procedural links* among them

3.42
Object-Process Language
OPL
subset of English natural language that represents textually the *OPM* model that the *OPD* represents graphically

3.43
Object-Process Methodology
OPM
formal language and method for specifying complex, multidisciplinary *systems* in a single function-structure-behaviour unifying model that uses a bimodal graphic-text representation of *objects* in the *system* and their *transformation* or use by *processes*

3.44
OPD object tree
tree graph, whose root is an *object*, depicting elaboration of the *object* through *refinement*

3.45
OPD process tree
tree graph whose root is the *System Diagram (SD*) and each node is an *OPD* obtained by in-*zooming* of a *process* in its ancestor *OPD* (or the *SD*) and each directed edge points from the *in-zoomed process* at the parent *OPD* to the same *process* in the child *OPD*

NOTE OPM model elaboration usually occurs by process decomposition through in-zooming, therefore the OPD process tree is the primary way to navigate an OPM model.

3.46
operation
process that a *thing* performs, which characterizes the *thing* other than itself

3.47
output link
link from the transforming *process* to the output (destination) *state* of an *object*

3.48
out-zooming
<object> inverse of *object in-zooming*

3.49
out-zooming
<process> inverse of *process in-zooming*

3.50
perseverance
property of *thing* which can be static, defining an *object*, or dynamic, defining a *process*

3.51
postcondition
<process> condition that is the outcome of successful process completion

3.52
postprocess object set
collection of *objects* remaining or resulting from *process* completion

NOTE The postprocess object set may include stateful objects, for which specific states result from process performance.

3.53
precondition
<process> condition for starting a *process*

3.54
preprocess object set
collection of *objects* to evaluate prior to starting a *process*

NOTE The collection of the *objects* may include *stateful objects* for which specific *states* are necessary for *process* performance.

3.55
primary essence
<system> *essence* of the majority of *things* in a system, which can be either *informatical* or physical

3.56
procedural link
graphical notation of *procedural relation* in *OPM*

3.57
procedural relation
connection or association between an *object* or *object state* and a *process*

NOTE 1 Procedural relations specify how the system operates to attain its function, designating time-dependent or conditional initiating of processes that transform objects.

NOTE 2 An invocation or exception link signifies a transient object in the flow of execution control between two processes.

3.58
process
transformation of one or more *objects* in the *system*

3.59
process class
pattern for *processes* that perform the same *object transformation* pattern

3.60
property
modelling annotation common to all *elements* of a specific kind that serve to distinguish that *element*

NOTE 1 Cardinality constraints, *path labels*, and structural link tags are frequent property annotations

NOTE 2 Unlike an attribute, the value of a property may not change during model simulation or operational implementation. Each kind of element has its own set of properties.

NOTE 3 Property is an attribute of an element in the OPM metamodel.

3.61
refineable, noun
<OPM> *thing* amenable to refinement, which can be a whole, an exhibitor, a general, or a class

3.62
refinee
thing that refines a *refineable*, which can be a part, a *feature*, a specialization, or an *instance*

NOTE Each of the four kinds of refinees has a corresponding refineable.(part-whole, feature-exhibitor, specialization-generalization, instance-class)

3.63
refinement
<model> elaboration that increases the extent of detail and the consequent model *completeness*

3.64
resultee

transformee that a *process* occurrence creates

3.65
stakeholder
<OPM> individual, organization, or group of people that has an interest in, or might be affected by the *system* being contemplated, developed, or deployed

3.66
stateful object
object with specified *states*

3.67
stateless object
object lacking specified *states*

3.68
state
<object> possible situation or position of an *object*

NOTE In OPM there is no concept of *process state*, such as "started", "in process", or "finished" within a model. Instead, OPM represents and models subprocesses, such as starting, processing, or finishing. Also see discussion of OPM process metamodel in Annex C.

3.69
state
<system> snapshot of the *system* model taken at a certain point in time, which shows all the existing *object instances*, current *states* of each *stateful object instance*, and the *process instances,* with their elapsed times, executing at the time the snapshot occurs

3.70
state expression
refinement involving the revealing of any proper subset of an *object's* set of *states*

3.71
state suppression
abstraction involving the hiding of any proper subset of an *object's* set of *states*

3.72
structural link
graphic notation of *structural relation* in *OPM*

3.73
structural relation
operationally invariant connection or association between things

NOTE Structural relations persist in the system for at least some interval of time. They provide the structural aspect of the system, and are not contingent upon conditions that are time-dependent.

3.74
structure
<OPM> collection of *objects* in an OPM model and the non-transient relations or associations among them

3.75
System Diagram
SD
OPD with one systmeic *process* indicating the *system function* and the *objects* connecting with that *function* to depict the overall context for and top-level view of the *system*

NOTE SD is the root of the OPD process tree and has no extent of detail beyond the overall context depicted, i.e. no in-zoomed refinee is present. Any OPD other than SD is a node in the OPD process tree resulting from refinement.

3.76
thing
<OPM> *object* or *process*

3.77
transformation
creation (generation, construction) or consumption (elimination, destruction) of an *object* or a change in the *state* of an *object*

NOTE Only a process can perform transformation.

3.78
transformee
object that a *process* transforms (creates, consumes, or affects)

3.79
transforming link
consumption link, effect link, or result link

3.80
unfolding
refinement that elaborates a refinee with additional detail comprising other *things* and the *links* between them.

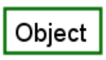
NOTE 1 The four kinds of unfolding are part unfolding, feature unfolding, specialization unfolding, and instance unfolding

NOTE 2 Unfolding is primarily applied to objects for exposing details about the unfolded object.

3.81
value
<attribute> *state* of an *attribute*

3.82
value
<functional> benefit at cost that the *system's function* delivers

4 Symbols



object

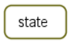
physical object

environmental object

process

physical process

environmental process



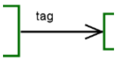
state


aggregation-participation


exhibition-characterization

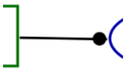

generalization-specialization

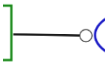

classification-instantiation



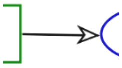
unidirectional tagged structural link

bidirectional tagged structural link

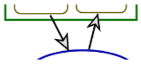

link

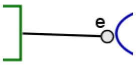

link


effect link

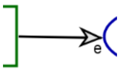

consumption link


result link

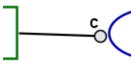

input-output link pair



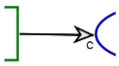
instrumental event link



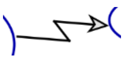
consumption event link



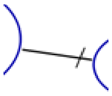
instrumental condition link

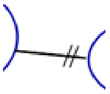


consumption condition link


invocation link

 self-invocation link

 over-time exception

 under-time exception

5 Conformance

Anticipating that the implementation of this Publically Available Specification by toolmakers and utilization by end-users is likely to occur in increments over time, several kinds of conformance criteria are appropriate.

a) Partial (symbolic) conformance with Object-Process Methodology, through utilizing the language part of Object-Process Methodology, namely OPM Semantics and Syntax:

   1) using only OPM symbols defined in Clause 4 of this document with the meaning assigned to them in this document; and,

   2) using only OPM elements defined in Clause 7 through Clause 12 of this document with the meaning assigned to them in this document.

b) Full conformance with Object-Process Methodology:

   1) conformance with (a) above; and,

   2) conformance with the approach and scheme of modelling systems with OPM, as defined in Clause 6 and Clause 14 of this document.

c) Conformance by toolmakers:

   1) conformance with (a) above;

   2) provision for (b) – users are guided and helped to adhere to (b) on the basis of the formalism of (a); and,

   3) support for OPL according to the EBNF definition specified in **Error! Reference source not found.** of his document.

6 Object-Process Methodology principles and concepts

6.1 OPM modelling principles

6.1.1 Modelling as a purpose-serving activity

System function and modelling purpose shall guide the scope and extent of detail of an OPM model. A complex or complicated system may involve many stakeholders, including the beneficiary, owner, users, and regulators, as well as many hardware and software components, exposing different aspects relevant to each stakeholder. The function or benefit expectations of stakeholders in general and beneficiaries in particular shall identify and prescribe the modelling purpose. This, in turn, shall determine the scope of the system model.

EXAMPLE For a manufacturing plant that produces widgets, the viewpoint of the marketing manager, who cares about supply rates and dates, does not include the machines in the plant that are used as instruments for making widgets, which are not affected by the marketing process. However, from the viewpoint of the maintenance manager, the machines definitely are affected as they become worn during operation and need to be maintained, both to prevent them from breaking and to fix them when they do break. Therefore, the OPM manufacturing plant model for the marketing manager will differ substantially from that constructed for the maintenance manager.

6.1.2 Unification of function, structure, and behaviour

The OPM structure model of a system shall be an assembly of the physical and informatical (logical) objects connected by structural relations. During the lifetime of a system, creation and destruction of those structural relations may occur.

The OPM behaviour model of a system, referred to as its dynamics, shall reflect the mechanisms that act on the system over time to transform systemic objects, i.e. objects that are internal to the system, and/or environmental objects, i.e. objects that are external to the system.

The combination of system structure and behaviour enables the system to perform a function, which shall deliver the (functional) value of the system to at least one stakeholder, who is the system's beneficiary. An OPM model integrates the functional (utilitarian), structural (static), and behavioural (dynamic) aspects of a system into a single, unified model. Maintaining focus from the viewpoint of overall system function, this structure-behaviour unification provides a coherent single frame of reference for understanding the system of interest, enhancing its intuitive comprehension while adhering to formal syntax.

6.1.3 Identify functional value

The functional value providing process of a modelled system shall express the function of the system as perceived by the system's main beneficiary or beneficiaries group. Identifying and labelling this primary process, the system's function, is a critical first step in constructing an OPM model according to the methodology prescription of the Object-Process Methodology approach. An appropriate function label or name should clarify and emphasize the central goal of the modelled system and the functional value that the system should provide for its main beneficiary. Modelling with OPM should begin by defining, naming, and depicting the function of the system as its primary process.

NOTE Such a deliberation, which often provokes a debate between the system architecture team members at this early stage, is extremely useful, as it exposes differences and often even misconceptions among the participants regarding the system which they set out to architect, model, and design.

After the function of the system aligns with the functional value expectation of its main beneficiary, the modeller shall identify and add other principal stakeholders to the OPM model.

6.1.4 Function versus behaviour

The value of the function to the beneficiary is often implied and expressed in process terms, which emphasize what happens, the behaviour, rather than the purpose, the functional value, for which the primary process happens. The modeller should distinguish between function and behaviour to create a clear and unambiguous system model. This distinction is essential because in many situations a system's function is achievable by different concepts, each implementing a different design and behaving differently.

EXAMPLE Consider a system for enabling humans to cross a river with their vehicles. Two obvious concepts are a static structure to enable car crossing and a dynamic moving element carrying cars. The corresponding system designs are a bridge and a ferry. While the function and the primary process – **River Crossing** – are identical for both designs, they differ dramatically in their structure and behaviour.

Failure to recognize the difference between function and behaviour may lead to a premature choice of a sub-optimal design. In the example above, this could result in making a decision to build a bridge without considering the possibly superior ferry option at all.

6.1.5 System boundary setting

The system's environment shall be a collection of things, which are outside of the system but which may interact with the system, possibly changing the system and its environment. The modeller shall distinguish these environmental things, which are not part of the system, from systemic things, which are part of the system. The modeller is not able to architect, design or manipulate the structure and behaviour of environmental things even though those environmental things may influence or be influenced by the system.

6.1.6 Clarity and completeness trade-off

Overwhelming detail and complicatedness are inherent in real-life systems. Making such systems understandable entails a trade-off that should balance between two conflicting criteria: clarity and completeness. Clarity shall be the extent of unambiguous comprehension that the system’s structure and behaviour models convey. Completeness shall be the extent of specification for all the system's details. These two model attributes conflict with each other. On the one hand, completeness requires the full stipulation of system details. On the other hand, the need for clarity imposes an upper limit on the extent of detail within an individual model diagram, after which comprehension deteriorates because of clutter and overloading.

Establishing an appropriate balance requires careful management of context during model development. The increase in the expression of completeness in a given model diagram often results in the reduction of clarity. However, the modeller may take advantage of the union of information provided by the entire OPM system model and have one diagram which is clear and unambiguous but not complete, and another that focuses on completeness for some portion of the system with more detail.

6.2 OPM Fundamental concepts

6.2.1 Bimodal representation

An OPM model shall be bimodal with expression in semantically equivalent graphics and text representations. Each OPM model graphical diagram, i.e. an Object-Process Diagram (OPD), shall have an equivalent OPM textual paragraph comprised of one or more OPM language sentences using the Object-Process Language (OPL).

NOTE 1 The bimodal graphics-text representation of the OPM model helps to involve non-technical stakeholders in the requirements elicitation and initial conceptual modelling of the system under development. This involvement engages those stakeholders as active participants and helps detect errors soon after their inadvertent introduction. The bimodal representation also helps novice OPM users quickly gain familiarity with the semantics of the OPM graphic modality when inspecting the text and corresponding graphic in tandem.

NOTE 2 Annex A specifies the OPL syntax using the conventions of ISO/IEC 14977.

NOTE 3 For most of the OPD figures throughout this International Standard, the corresponding paragraph of OPL sentences accompanies the graphical OPD.

6.2.2 OPM modelling elements

Elements, the basic building blocks of any system modelled in the Object-Process Methodology (OPM), shall be of two kinds: things and links. The modelling elements of object and process shall designate things in the model context. The modelling element of link shall designate associations between things in the model context. Objects shall be stateless or have object states. Links shall be either procedural or structural.

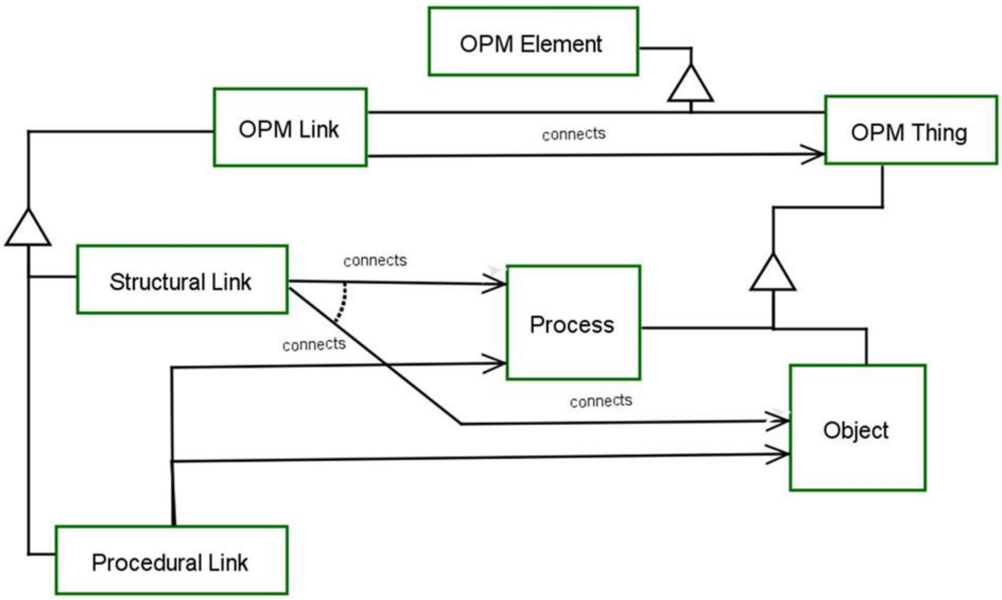


Figure 1 — OPM metamodel overview

Within an OPM model, modelling elements shall have unique symbols, textual expression, syntactic constraints and semantic interpretation. Within an OPM model, each modelled thing shall have a unique identifying name of relevance to model stakeholders and unique source and destination things shall distinguish each link or tagged link. A modelled link, together with its source and destination things shall be an OPM construct that has a corresponding OPL sentence.

Once identified, a modelled thing may appear in any relevant context for that thing and may appear more than once in a context to enhance understanding.

6.2.3 OPM things: objects and processes

An object shall be a thing, which, once constructed, exists or can exist physically or informatically. Associations among objects shall constitute the object structure of the system being modelled, i.e. the static, structural aspect of the system. An object state shall be a particular situational classification of an object at some point during its lifetime. At every point in time, an object with an object state is in one of its states or in transition between two of its states as a consequence of a process currently affecting that object.

A process shall be a thing that expresses the transformation of objects in the system. A process is always associated with and occurs or happens to one or more objects; it does not exist in isolation. A process transforms objects by creating them, consuming them, or changing their state. Thus, processes complement objects by providing the dynamic, behavioural aspect of the system.

NOTE Inspecting processes to determine which subprocess is performing at the point in time of inspection reveals the status of a process. OPM does not specify explicitly the model state of a process. See process metamodel in Annex C.

6.2.4 OPM links: procedural and structural

Procedural links shall denote procedural relations. A procedural relation shall specify how the system operates to attain its function, designating time-dependent or conditional initiating of processes, which transform objects.

Structural links shall denote structural relations. A structural relation shall specify an association that persists in the system for at least some interval of time, i.e. a static aspect of the system, and shall not be contingent upon conditions that are time-dependent.

6.2.5 OPM context management

OPM shall provide mechanisms for managing the contextual scope of model detail to promote both comprehension and clarity. From the initial functional model context, the modeller shall use refinement of object and process structure to extend model detail with each incremental extent of detail comprising a contextual focus.

To achieve the system function, a set of non-trivial processes shall comprise a hierarchical network of sub-processes. The process hierarchy shall induce a partial order on the processes, i.e. some processes end before others can start, while other processes may occur in parallel or as alternatives. At any extent of detail in the process hierarchy, a process in a system should provide or contribute functional value as part of its ancestor process.

The fundamental unit of context management is the Object-Process Diagram (OPD) that depicts the modelling elements of that particular context. New diagram unfolding and new diagram in-zooming provide structural and procedural connections between contexts. Although any OPD may include any number of elements, only those elements pertinent to the particular context should appear in the OPD.

The management context for names and labels of things and links shall be the entire OPM model for which separate model fragments contextualize the relationships and interactions among model elements that produce behaviour. Relations to their refineables disambiguate identical names for different things.

6.2.6 OPM model implementation (informative)

6.2.6.1 Conceptual models versus runtime models

When constructing models with OPM, modellers need to understand the distinction between the conceptual model they are creating and an operational occurrence of that model that they may use to assess system behaviour. Practicing modellers have an intuitive sense for this distinction, readily thinking of modelling element operational instance occurrences when creating a model, even when those elements are very abstract. However, those not familiar with modelling of the kind OPM supports may find the specification of this Publically Available Specification somewhat confusing.

An OPM model is a formal framework within which object and process occurrences interact by means of links. Because an OPM model has this kind of framework, akin to the system's structure, and model elements interact using links, the modeller may simulate system behaviour by creating object and process operational instance occurrences, and then follow the flow of execution control embodied in the connections and OPM semantic rules. The presence of thing occurrences translates the abstract conceptual model into a more concrete runtime form.

Annex D presents OPM facilities to support simulation activities. However, as the users of this Publically Available Specification construct OPM models, they need to keep in mind that the behaviour of the modelled system occurs only when operational instance occurrences of things exist. The appearance of a link between two things does not imply behaviour until operational instance occurrences of those things exist. The word 'runtime', i.e. when operational instance occurrences do exist, is implicit in every specification statement provided herein.

NOTE The word 'instance' also occurs with a different meaning in the presentation of the classification-instantiation relation. In that usage, an instance is a refinee typical of the class.

6.2.6.2 OPM model realization

The conceptual framework for OPM includes the capability for model simulation. To use this capability successfully, a modeller needs to understand the distinction between a model as a representation of a pattern of structure and behaviour and an instance of the model operating to perform the function for which the model is a pattern. The model has an architectural form, based in part on the arrangement of structure and procedure, which the modeller extends with detail as the model design evolves. A model expressing consistent detail is implementable as a simulation, i.e. capable of realizing resources, using processes to transform objects, and to produce functional value to a beneficiary.

6.2.6.3 OPD Navigation and OPL composition

This Publicly Available Specification expresses the means for creating OPM model diagrams and corresponding OPL texts. The in-zooming and unfolding mechanisms of Clause 14 provide ways to link OPD diagrams with corresponding OPL to express the linkage as text. However, because there are many ways to label these links, some of which may be specific to a tool implementation, Clause 14 does not specify the labels to assign for identifying successive hierarchic levels, linkage between related OPD diagrams, or OPL segments.

7 OPM thing syntax and semantics

7.1 Objects

7.1.1 Description

An object shall be a thing that exists or has the potential of physical or informatical existence. From the temporal viewpoint, the existence of an object shall be persistent. As long as no process acts on the object, it shall remain in its current implicit or explicit state.

An OPM object is an abstract category identifier for a pattern of structure, properties and features, i.e. attributes and operations, that are applicable to operational instance objects of that category. Within constraints of the model, any non-negative number of object operational instances may exist.

7.1.2 Representation

A rectangular box containing a label, the object name, shall signify graphically the presence of a model object. Figure 2 graphically illustrates the object **Vehicle Occupant Group**. In OPL text, the object name shall appear in bold face with capitalization of each word.

Figure 2 — Object graphic notation

NOTE Sub-clause B.6.2 discusses conventions for naming objects.

7.2 Processes

7.2.1 Description

A process shall be a thing that transforms one or more objects. Transformation may be generation (construction, creation), effect, or consumption (destruction, elimination). A process shall have positive performance time duration.

An OPM process is an abstract category identifier for a pattern of transformation. For the concrete, operational instance realization, a process instance is a specific occurrence of the process pattern that the category specifies. The process operational instance transforms one or more object operational instances.

NOTE 1 A process may directly invoke another process, by means of the invocation link (see sub-clause 9.5.2.5.2), which results in the invoking process creating a transient object that the invoked process immediately consumes.

NOTE 2 The effect of a process on an object is usually a change in that object's state. However, there are persistent processes whose effect is state maintenance. Rather than inducing a change, the semantics of a persistent process is to leave the object in a steady state by leaving the object in its current state.

EXAMPLE The process **Existing** is the most prominent persistent process; it describes a static (implicit) state of existence. Examples of other persistent processes are **Holding**, **Maintaining**, **Keeping**, **Staying**, **Waiting**, **Prolonging**,

Extending, **Delaying**, **Occupying**, **Persisting**, **Continuing**, **Supporting**, **Withholding**, and **Remaining**. For biological objects, **Existing** entails **Living** – actively maintaining the necessary life processes.

7.2.2 Representation

An ellipse containing a label, the process name, shall signify graphically the presence of the abstract process category. Figure 3 graphically illustrates the process **Automatic Crash Responding.** In OPL text, the process name shall appear in bold face with capitalization of each word.

Figure 3 — Process graphic notation

NOTE Sub-clause B.6.3 discusses conventions for naming processes.

7.3 OPM things

7.3.1 OPM thing defined

An OPM thing shall be an object or a process. Objects and processes are symmetric in many regards and have much in common in terms of relations, such as aggregation, generalization and characterization. An object exists while a process happens to one or more objects. OPM objects and OPM processes depend on each other in the sense that a process is necessary to transform an object, while at least one object to transform is necessary for a process to occur or happen.

7.3.2 Object-process test

To apply OPM in a useful manner, the modeller needs to make the essential distinction between objects and processes, as a prerequisite for successful system analysis and design. By default, a noun shall identify an object. The object-process test provides modellers with criteria to distinguish nouns used for processes from nouns used for objects. Providing a correct answer to the question about whether a given noun is an object or a process is crucial and fundamental to object-process methodology.

To be a process, a noun or noun phrase shall satisfy each of the following three process criteria:

– time association, the noun in question associates with the passage of time;

– verb association, the noun in question derives from, or has a common root with a verb, or has a synonym that associates with a verb; and

– object transformation, the noun in question occurs, happens, performs, executes, transforms, changes, or alters at least one object, or maintains it in its current state.

EXAMPLE **Flight** is a noun that is a process because it passes all three object-process test criteria: 1) it has a time association; 2) it associates with the verb to fly; and 3) it transforms **Airplane** by changing the value of its location attribute from source to destination.

7.3.3 OPM thing generic properties

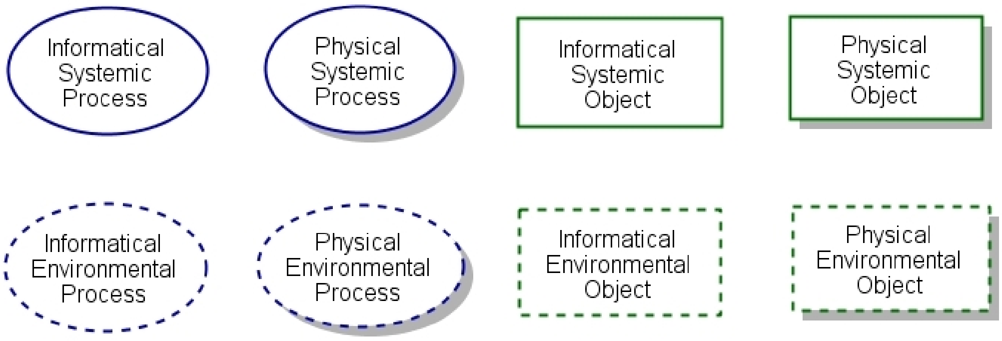
All OPM things shall have the following three generic properties:

– **Perseverance**, which pertains to the thing's persistence and denotes whether the thing is static, i.e. an object, or dynamic, i.e. a process. While objects are persistent, i.e. they have static perseverance, and processes are transient, i.e. they have dynamic perseverance, boundary examples of persistent processes (see 7.2.1), as well as of transient objects (see sub-clause 9.5.2), may exist. Accordingly, the permissible value for the **Perseverance** property is static, dynamic or persistent.

– **Essence**, which pertains to the thing's nature and denotes whether the thing is physical or informatical. Accordingly, the permissible value of the generic attribute **Essence** is physical or informatical.

– **Affiliation**, which pertains to the thing's scope and denotes whether the thing is systemic, i.e. part of the system, or environmental, i.e. part of the system's environment. Accordingly, the value of the property **Affiliation** is systemic or environmental.

Graphically, as shown in Figure 4, shading effects shall denote physical OPM things and dashed lines shall denote environmental OPM things. All eight **Perseverance**-**Essence**-**Affiliation** generic property combinations of an OPM thing shown in Figure 4 may occur. The lower portion of Figure 4 expresses, from left to right and top to bottom, the OPL sentences corresponding to the graphical elements.



Informatical Systemic Process is an informatical and systemic process.
Physical Systemic Process is a physical and systemic process.
Informatical Systemic Object is an informatical and systemic object.
Physical Systemic Object is a physical and systemic object.
Informatical Environmental Process is an informatical and environmental process.
Physical Environmental Process is a physical and environmental process.
Informatical Environmental Object is an informatical and environmental object.
Physical Environmental Object is a physical and environmental object.

Figure 4 — OPM thing generic attribute combinations

7.3.4 Default values of thing generic properties

The default value of the Affiliation generic property of a thing shall be systemic.

Any non-trivial system tends to have a majority of objects and processes with the same thing generic property values for Essence.

EXAMPLE Data processing systems are informatical, although they have physical components. A transportation system, such as a railway system or an aviation system, is physical, although they have informatical components.

A system's Primary Essence shall be the same as that of the majority thing Essence values within the system boundary.

The default value of the Essence generic property of a thing within the boundary of a system shall be the Primary Essence of the system.

NOTE A supporting tool should provide an option for the modeller to specify a system's Primary Essence as a means to establish the default thing generic attribute value for Essence.

The OPL corresponding to a diagram shall not reflect the default values of thing generic properties unless the thing does not yet connect to another thing, e.g., during the course of the modelling process. As soon as links to other things appear, thing generic properties shall merge as appropriate into OPL phrases describing these links.

7.3.5 Object states

7.3.5.1 Stateful and stateless objects

Object state shall be a possible situation in which an object may exist. An object state has meaning only in the context of the object to which it belongs, i.e. the object that has the state.

A stateless object shall be an object that has no specification of states.

A stateful object shall be an object with a specified set of permissible states. In a runtime model, at any point in time, any stateful object operational instance is at a particular permissible state or exists in transition between two permissible states as a consequence of a process currently affecting that object.

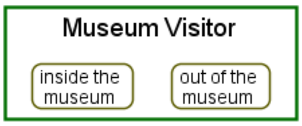
NOTE 1 Depending upon model behaviour, operational instances of an object may be at different states.

NOTE 2 Sub-clause B.6.4 discusses conventions for naming object states.

7.3.5.2 Object state representation

Graphically, a labelled, rounded-corner rectangle (a 'rountangle') placed inside the object to which it belongs shall denote an object state. In OPL text, the object state label shall appear in bold face without capitalization.

EXAMPLE Figure 5 depicts the object **Museum Visitor** with two states labelled **inside the museum** and **out of the museum**. Below the graphical representation is the corresponding OPL sentence.



Museum Visitor can be **inside the museum** or **out of the museum**.

Figure 5 — A stateful object with two states

7.3.5.3 Initial, default, and final states

The initial state of an object shall be its state as the system begins operating or its state upon generation by the system during operation. The final state of an object shall be its state as the system completes operation or its state upon consumption by the system during operation. The default state of an object shall be the state in which the object is most likely to be upon random inspection.

NOTE 1 An object may have zero or more initial states, zero or more final states, and zero or one default state. The same state can be any combination of initial, final and/or default.

NOTE 2 The initial and final states are especially useful for objects that exhibit a lifecycle pattern, such as a product or an organism or a system.

NOTE 3 If an object has more than one initial state, then it is possible to assign to each initial state a probability of the object being created in that state (see 12.7).

7.3.5.4 Initial, default, and final state representation

Graphically, a thick contour border shall denote an initial state, a double contour border shall denote a final state, and an open arrow pointing diagonally from the left shall denote a default state. The corresponding OPL sentences make the state specification explicit.

EXAMPLE Figure 6 depicts the object **Specification** with initial, default and final states. Below the graphical representation are the corresponding OPL sentences.

State **preliminary** of **Specification** is initial**.**
State **approved** of **Specification** is default**.**
State **cancelled** of **Specification** is final**.**

Figure 6 — A stateful object with initial, default, and final states

7.3.5.5 Attribute values

Since an attribute is an object, an attribute value shall correspond to a state in the sense that a value is a state of an attribute. An object may have an attribute, which is a different object, and for some time interval during the existence of the object exhibiting that attribute, the value of that attribute is the state of the different object.

EXAMPLE Considering **Temperature in degrees Celsius** as an attribute of **Engine**, **75** is a value of that attribute.

NOTE 1 Since an attribute is a stateful object, a permissible attribute value is a member of the set of permissible states of that stateful object. An enumerated list or a set of one or more ranges of numbers may define the set of permissible values for the attribute.

NOTE 2 In contrast, a property value is fixed and does not change during model operation.

Attributes with values expressed in measurement units shall express the measurement unit graphically in an OPD within brackets below the attribute object name and express the measurement unit in text after the attribute object name in corresponding OPL sentences, e.g., **Temperature** in **degrees Celsius**.

8 OPM link syntax and semantics overview

8.1 Procedural link overview

8.1.1 Kinds of procedural links

A procedural link shall be one of three kinds:

— Transforming link, which connects a transformee (an object that the process transforms) or one of its states with a process to model object transformation, namely generation, consumption, or state change of that object as a result of the process performance;

— Enabling link, which connects an enabler (an object that enables the process occurrence but is not transformed by that process), i.e. an agent or an instrument, or its state, with a process to model an enabling presence for that process; or

— Control link, which is a transforming or an enabling link with the added semantics of an execution control mechanism to model an event that initiates a linked process, to model a condition for process performance, or to model a connection of two processes denoting invocation, or exception.

NOTE Transformee and enabler are roles an object may have with respect to the process to which they link. Hence, an object may have the role of an enabler for one process and a transformee for another process.

8.1.2 Procedural link uniqueness OPM principle

A process shall connect with a transforming link to at least one object or object state. At any particular extent of abstraction, an object or any one of its states shall have exactly one role as a model element with respect to a process to which it links: the object may be a transformee, an enabler, an initiator, or a conditional object. At a given extent of abstraction, an object or an object state shall link to a process by only one procedural link.

8.1.3 State-specified procedural links

Each procedural link may be qualified as a state-specified procedural link. A state-specified procedural link shall be a procedural link that connects a process to a specified state of an object.

8.2 Operational semantics and flow of execution control

8.2.1 The Event-Condition-Action control mechanism

The Event-Condition-Action paradigm shall provide the OPM operational semantics and flow of execution control. At the point in time of object creation, or appearance of the object from the system's perspective, or entrance of an object to a particular state, an event shall occur. At runtime, for objects that are the source of a link with a process, e.g. enabler of a process, the occurrence of an event shall initiate evaluation of the precondition for every process to which the object links as a link source.

When the precondition evaluation for a process begins, the event shall cease to exist for that process. If and only if the evaluation reveals satisfaction of the precondition shall the process start performance of the process and action occurs.

Starting performance of a process has two prerequisites: 1) an initiating event, and 2) satisfaction of a precondition. Thus, events and preconditions in concert specify OPM flow of execution control for process performance.

NOTE Invocation and exception are event-condition-actions that occur only between processes.

The flow of execution control shall be the consequence of successive Event-Condition-Action sequences that begin with initiation of the system function by an external event and end when the system function is complete.

8.2.2 Preprocess object set and postprocess object set

The preprocess object set of a process shall determine the precondition to satisfy before performance of that process starts. The preprocess object set may be complex and include compound logical expressions, or may simply include the existence of one or more objects, possibly in specified states. Typical objects in a preprocess object set are consumees, i.e. objects the process consumes, affectees, i.e. objects the process affects, and process enablers. Some of these objects may have a further stipulation regarding flow of execution control, i.e. a condition link. Every process shall have a preprocess object set with at least one object, possibly in a specified state.

The postprocess object set shall determine the postcondition that process completion satisfies. The postprocess object set may be complex and include compound logical expressions, or may simply include the existence of one of more objects, possibly in specified states. Typical objects in a postprocess object set are resultees, i.e. objects the process generates and affectees, i.e. objects the process affects. Every process shall have a postprocess object set with at least one object, possibly in a specified state.

NOTE 1 The intersection of the preprocess object set and the postprocess object set of the same process includes the process enablers and affectees. Consumees are only members of the preprocess object set, while resultees are only members of the postprocess object set.

NOTE 2 Clause 14.2.2.4.4 presents the operational instance semantics for objects in the involved object set.

8.2.3 Skip semantics of condition vs. wait semantics of non-condition links

A process preprocess object set may include both condition links (see 9.5.3) and non-condition links, i.e. procedural links without the condition control modifier. The distinguishing aspect of condition links is their 'skip semantics', which provide for skipping or bypassing a process if the source object operational instance of the condition link does not exist. Without the condition link qualification, the non-existence of a source object operational instance causes the process to wait for another event and operational instances of all source objects to exist, possibly in a specified state, thus satisfying the precondition.

If there are one or more non-condition links and one or more condition links, the existence of all of them shall be necessary to satisfy the precondition and start the process. However, if there are one or more unsatisfied non-condition links and one or more unsatisfied condition links, a conflict arises between the wait semantics of the former and the skip semantics of the latter. To resolve the conflict, the skip semantics of the condition links shall be stronger than the wait semantics of their non-condition counterparts and the flow of execution control bypasses the process, which does not start its performance or generate an exception.

Even if just one of the conditions attendant to the condition links connecting with the process does not exist, the precondition satisfaction evaluation shall fail, execution control skips the process, and an event occurs for the next sequential process(es) by means of an invocation link of some kind, see 9.5.2.5 and 14.2.2.

NOTE 1 There is no result event link or result condition link, because these are outgoing procedural links relating to the postprocess object set. When a process completes, it creates the postprocess object set without further condition, so there is no condition on the creation of resultees or change of affectees. Creation of an object, possibly at a specified state, in the postprocess object set may serve as an event or condition for the next sequential process(es).

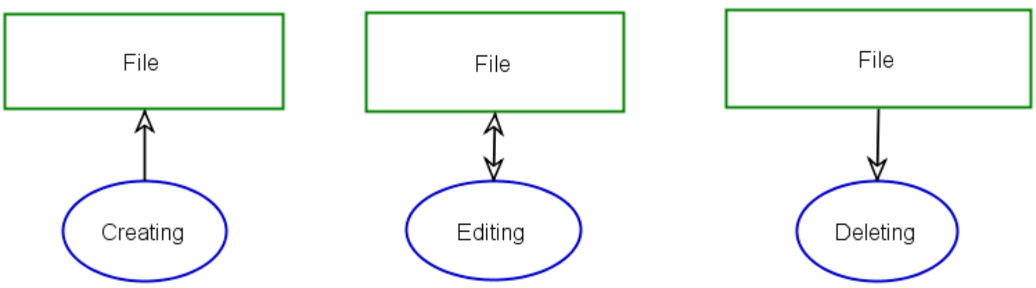
NOTE 2 To achieve robust flow of execution control under all circumstances, the modeller should model premature process ending without completion as exception handling (see 9.5.4).

9 Procedural links

9.1 Transforming links

9.1.1 Kinds of transforming links

A transforming link shall specify a connection between a process and its transformee (the object it consumes, creates, or changes the object state). The three kinds of transforming links shall be consumption link, result link, and effect link. Figure 7 illustrates the three kinds of transforming connections with the corresponding OPL sentences below the graphical representation.



Creating yields **File** **Editing** affects **File** **Deleting** consumes **File**

Figure 7 — Transforming links: left – result, middle – effect, right – consumption

A transformee shall be a role that an object has with respect to a given process. The same object may have a different role for another process.

9.1.2 Consumption link

A consumption link shall be a transforming link specifying that the linked process consumes (destroys, eliminates) the linked object, the consumee.

Graphically, an arrow with a closed arrowhead, as shown in Figure 7, pointing from the consumee to the consuming process shall denote the consumption link.

The syntax of a consumption link OPL sentence shall be: **Processing** consumes **Consumee.**

Existence of the consumee shall be a precondition, or part of the precondition, for process activation. If the consumee does not exist, i.e. no operational instance of the consumee exists, then process activation shall wait for the consumee to exist.

The consumption shall be immediate upon process activation, unless the modeller needs to model consumption of the object over time. In this case, the consumption link shall have a property that indicates the rate of consumption of the consumee and the consumee shall have an attribute that indicates the available quantity.

NOTE 1 The modeller may create an exception if the object quantity is less than the rate times the expected process duration.

NOTE 2 See 11.1 for the denotation of link properties.

EXAMPLE 1 **Steel Rod** is a consumee for the process **Machining**, which generates the resultee **Shaft**. Once **Machining** has started, it consumes **Steel Rod**.

EXAMPLE 2 **Water** is a consumee for the process **Irrigating**. The consumee has an attribute **Quantity [liter]** with value **1000** and the consumption link has a property **Flow Rate [liter/sec]** with value **50**. In this case, if **Irrigating** is uninterrupted, it will last 20 seconds, and it will consume **Water** at the specified **Flow Rate** value.

9.1.3 Result link

A result link shall be a transforming link specifying that the linked process creates (generates, yields) the linked object, which is the resultee.

Graphically, an arrow with a closed arrowhead, as shown in Figure 7, pointing from the creating process to the resultee shall denote a result link.

The syntax of a result link OPL sentence shall be: **Processing** yields **Resultee.**

The generation of the resultee shall be immediate upon process completion, unless the modeller needs to model the generation of the object over time. In this case, the result link shall have a property that indicates its rate of resultee generation and the resultee shall have an attribute that indicates the available quantity.

NOTE See 11.1 for the denotation of link properties.

EXAMPLE 1 **Steel Rod** is a consumee for the process **Machining**, which generates the resultee **Shaft**. When **Machining** completes, it generates **Shaft**.

EXAMPLE 2 **Gasoline** and **Diesel Oil** are resultees of the process **Refining**, which consumes **Crude Oil**. The resultees **Gasoline** and **Diesel Oil** each have an attribute **Quantity [cubic meter]**. The **Refining** to **Gasoline** result link has the property **Gasoline Yield Rate [cubic meter/hour]** with value **1000** and the **Refining** to **Diesel Oil** result link has the property **Diesel Oil Yield Rate [cubic meter/hour]** with value **800**. Assuming there is enough **Crude Oil**, if **Refining** activates and performs for 10 hours, it will yield 10,000 cubic meters of **Gasoline** and 8,000 cubic meters of **Crude Oil**.

9.1.4 Effect link

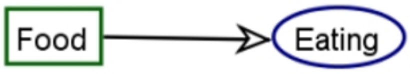
An effect link shall be a transforming link specifying that the linked process affects the linked object, which is the affectee, i.e. the process causes some unspecified change in the state of the affectee.

Graphically, a bidirectional arrow with two closed arrowheads, as shown in Figure 7, one pointing in each direction between the affecting process and the affected object shall denote the effect link.

The syntax of an effect link OPL sentence shall be: **Processing** affects **Affectee.**

9.1.5 Basic transforming links summary

Table 1 — Basic transforming links summary

Name	Semantics	Sample OPD & OPL	Source	Destination
Consumption link	The process consumes the object.	 **Eating** consumes **Food.**	consumed object	consuming process
Result link	The process generates the object.	 **Mining** yields **Copper.**	creating process	created object
Effect link	The process affects the object by changing it from one state to another state.	 **Purifying** affects **Copper.**	affected object and affecting process are both source and destination	

9.2 Enabling links

9.2.1 Kinds of enabling links

An enabling link shall be a procedural link specifying an enabler for a process. An enabler for a process shall be an object that is necessary for that process to occur. The existence and state of an enabler after the process is complete shall be the same as just before the process began its performance.

The two kinds of enabling links shall be agent link and instrument link.

The enabler shall be present throughout the performance of the process that it enables. If, from the system's viewpoint, the enabler ceases to exist during the performance of the process it enables, that process shall immediately end.

NOTE 1 An enabler is a role an object has with respect to a given process. The same object may be an enabler for one process and a transformee for another process.

NOTE 2 To achieve robust flow of execution control under all circumstances, the modeller should model premature process ending without completion as exception handling (see 9.5.4).

9.2.2 Agent and Agent Link

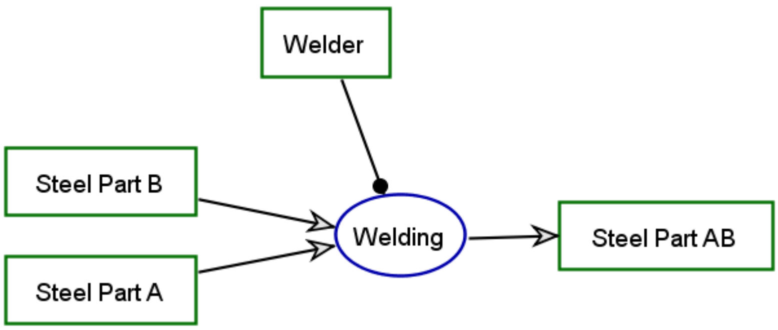
An agent shall be a human or a group of humans capable of intelligent decision-making, who interact with the system to enable or control the process throughout performance of the process.

An agent link shall be an enabling link from the agent object to the process it enables, specifying that the agent object is necessary for linked process activation and performance.

Graphically, a line with a filled circle resembling a black lollipop at the terminal end extending from the agent object to the process it enables shall denote an agent link.

The syntax of an agent link OPL sentence shall be: **Agent** handles **Processing.**

EXAMPLE 1 In the OPD in Figure 8, **Welder** is the agent for **Welding**. Performing the process of **Welding** the object **Steel Part A** with the object **Steel Part B** to create **Steel Part AB**, requires a human **Welder**. **Welder** is the agent of **Welding**. However, **Welding** does not transform the **Welder**, but **Welding** cannot take place without the **Welder**.



Welder handles **Welding.**
Welding consumes **Steel Part A** and **Steel Part B.**
Welding yields **Steel Part AB.**

Figure 8 — Agent link example

EXAMPLE 2 In the OPD in Figure 8, if, for whatever reason**, Welder** goes away before **Welding** completes, then **Welding** stops prematurely and the creation of **Steel Part AB** does not occur, although **Welding** already consumed **Steel Part A** and **Steel Part B**.

9.2.3 Instrument and Instrument Link

An instrument shall be an inanimate or otherwise non-decision-making enabler of a process that is not able to start or take place without the existence and availability of the instrument.

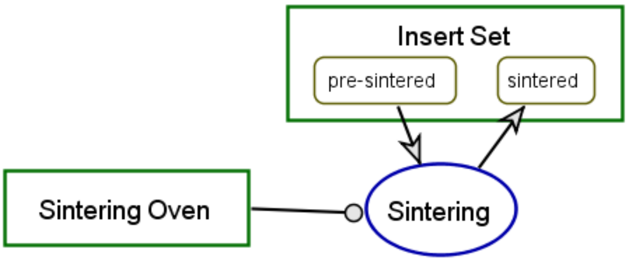
An instrument link shall be an enabling link from the instrument object to the process it enables, specifying that the instrument object is necessary for linked process activation and performance.

Graphically, a line with an open circle resembling a white lollipop at the terminal end extending from the instrument object to the process it enables shall denote an instrument link.

The syntax of an instrument link OPL sentence shall be: **Processing** requires **Instrument.**

EXAMPLE 1 A **Manufacturing** process may not consume or (disregarding wear and tear) change the state of a **Machine** that enables the transformation of **Bar Stock** to **Machined Part**. In this context, the **Machine** is an instrument of the **Manufacturing** process.

EXAMPLE 2 In the Figure 9 OPD, **Sintering Oven** is the instrument for **Insert Set**, because without it **Sintering** cannot happen. However, while the **Insert Set** object is transformed (its state changes from pre-sintered to sintered), disregarding wear and tear, **Sintering Oven** remains unaffected as a result of preforming the **Sintering** process.



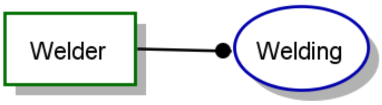
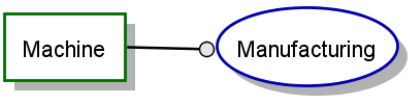
Insert Set can be **pre-sintered** or **sintered.**
Sintering requires **Sintering Oven.**
Sintering changes **Insert Set** from **pre-sintered** to **sintered.**

Figure 9 — Instrument link example

EXAMPLE 3 In the Figure 9 OPD, if during the **Sintering** process **Sintering Oven** ceases to exist, e.g., due to severe cracking, **Sintering** will stop and **Insert Set** will not be in its **sintered** state, although it already left its **pre-sintered** state.

9.2.4 Basic enabling links summary

Table 2 — Enabling links summary

Name	Semantics	Sample OPD & OPL	Source	Destination
Agent Link	Agent is a human or a group of humans who enables the occurrence of the process to which it is linked but is not transformed by that process.	 **Welder** handles **Welding.**	agent – the enabling object	enabled process
Instrument Link	Instrument is an inanimate object that enables the occurrence of the process to which it is linked but is not transformed by that process.	 **Manufacturing** requires **Machine.**	instrument – the enabling object	enabled process

9.3 State-specified transforming links

9.3.1 State-specified consumption link

A state-specified consumption link shall be a consumption link from a specified state of the consumee to the linked process that consumes (destroys, eliminates) the object. Existence of the consumee in the specified state shall be a precondition, or part of the precondition, for process activation. If the consumee is not in that specified state, then process activation shall wait for the consumee to exist at that specified state.

Graphically, an arrow with a closed arrowhead pointing from the specified state of the object to the process, which consumes the object, shall denote the state-specified consumption link.

The syntax of a state-specified consumption link OPL sentence shall be: **Process** consumes **specified-state Object.**

The consumption shall be immediate upon process activation, unless the modeller needs to model consumption of the object over time. In this case, the consumption link shall have a property that indicates the rate of consumption of the consumee and the consumee shall have an attribute that indicates the available quantity.

NOTE 1 The modeller may create an exception if the object quantity is less than the rate times the expected process duration.

NOTE 2 See 11.1 for the denotation of link properties.

EXAMPLE 1 **Steel Rod** at state **pre-heat-treated** is a consumee for the process **Machining**, which generates the resultee **Shaft**. When **Machining** activates, it consumes **pre-heat-treated Steel Rod**, because this **pre-heat-treated Steel Rod** is not available for any purpose other than becoming a **Shaft** resultee of this process. If **Steel Rod** previously went through a **Heat Treating** process, it is at state **heat-treated**, and therefore not available to undergo **Machining**.

EXAMPLE 2 Continuing with EXAMPLE 1, **Steel Rod** is at state **pre-heat-treated** and has an attribute **Quantity [units]** with value 600. The state-specified consumption link has a property **Rate [units/hour]** with value **60**. When **Machining** performs, it consumes the 600 Steel Rods after 10 working hours.

9.3.2 State-specified result link

A state-specified result link shall be a result link from a process to a specified state of the resultee object that the process creates (generates, yields). Existence of the resultee at the specified state shall be a postcondition, or part of the postcondition, upon completion of the generating process.

Graphically, an arrow with a closed arrowhead pointing from the process to the specified state of the object shall denote the state-specified result link.

The syntax of a state-specified result link OPL sentence shall be: **Process** yields **specified-state Object.**

The generation of the resultee at the particular state shall be immediate upon process completion, unless the modeller needs to model the generation of the object over time. In this case, the result link shall have a property that indicates its rate of resultee generation and the resultee shall have an attribute that indicates the available quantity at that specified state.

NOTE 1 See 11.1 for the denotation of link properties.

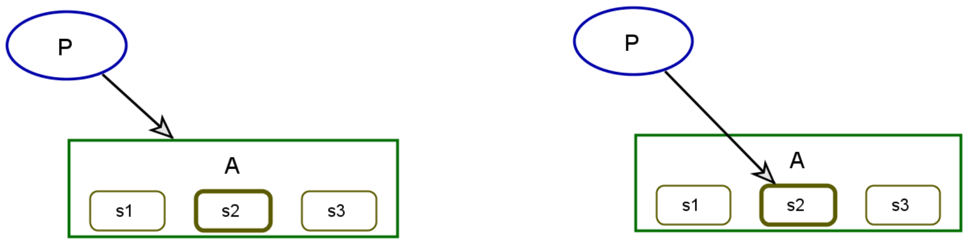
NOTE 2 At runtime an operating model may consist of multiple operational instances of an object with each operational instance at a different state.

EXAMPLE 1 **Steel Rod** at state **pre-heat-treated** is a consumee for the process **Machining**, which generates the resultee **Shaft** at state **pre-heat-treated**. A state-specified result link from **Machining** to the **pre-heat-treated** state of **Shaft** denotes this model specification.

A result link yielding a stateful object with an initial state should attach at that object rectangle or one of its states other than the initial state.

NOTE 3 The modeller may want the OPL on the right in Figure 10, but the OPL on the left reduces ambiguity.

EXAMPLE 2



A can be **s1, s2,** or **s3.**
S2 is initial**.**
P yields **A.**

A can be **s1, s2,** or **s3.**
S2 is initial**.**
P yields **s2 A.**

Figure 10 — Correct (left) and incorrect (right) result link to an object with an initial state

9.3.3 State-specified effect links

9.3.3.1 Input and output effect links

An input source link shall be the link from a specified state of an object, an input source, to the transforming process, while the output destination link shall be the link from the transforming process to a specified state of an object, an output destination. These links provide three possible modelling situations in the context of a single object linking to a single process: 1) input-output-specified effect link specifying both input source and output destination states; 2) input-specified effect link specifying only the input source state; and 3) output-specified effect link specifying only the output destination state.

9.3.3.2 Input-output-specified effect link

An input-output-specified effect link shall be a pair of effect links, where the input source link connects to an affecting process from a specified state of an affectee, and the output destination link connects from that same process to a different output destination state of the same affectee. Existence of the affectee at the input source state shall be a precondition, or part of the precondition, for affecting process activation. Existence of the affectee at the output destination state shall be a postcondition, or part of the postcondition, upon affecting process completion.

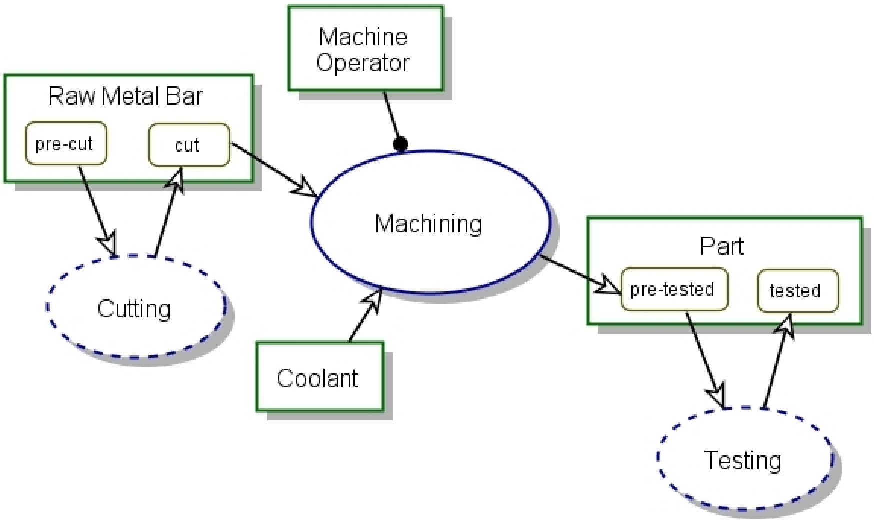
Graphically, a pair of arrows consisting of an arrow with a closed arrowhead from the input source state of the affectee to the affecting process, the input source link, and a similar arrow from that process to the output destination state of the affectee at process completion, the output destination link, shall denote the input-output-specified effect link.

The syntax of an input-output-specified effect link OPL sentence shall be: **Process** changes **Object** from **input-state** to **output-state.**

EXAMPLE 1 The OPD in Figure 11 depicts state-specified consumption and result links. **Machining** can only consume **Raw Metal Bar** in state **cut** and generate **Part** in state **pre-tested**. **Cutting** and **Testing** are environmental processes. **Cutting** must precede **Machining** in order to change **Raw Metal Bar** from its **pre-cut** to its **cut** state, while **Testing** changes **Part** from **pre-tested** to **tested**.

NOTE 1 In the case of an input-output-specified effect link, once an affecting process starts, it causes the object to exit out of its input source state. However, the object reaches its output destination state only when the process completes. Between process start and process completion, the affectee object is in transition between the two states.

EXAMPLE 2 In the OPD in Figure 11, **Cutting** takes **Raw Metal Bar** from its **pre-cut** to its **cut** state. As long as **Cutting** is active, the state of **Raw Metal Bar** is in transition and bound to the **Cutting** process: **Cutting** takes it out of its **pre-cut** state but has not yet brought it to its **cut** state with process completion. While **Cutting** the state of **Raw Metal Bar** is indeterminate: it could be partly cut and reusable or mostly cut and unusable. In either case, it is not available for **Machining**, since it is not in its **cut** state.



Raw Metal Bar is **physical.**
Raw Metal Bar can be **pre-cut** or **cut.**
Machine Operator is **physical**.
Coolant is **physical.**
Machining is **physical.**
Machining requires **Coolant.**
Machine Operator handles **Machining.**
Part is **physical.**
Part can be **pre-tested** or **tested.**
Testing is **environmental** and **physical.**
Cutting changes **Raw Metal Bar** from **pre-cut** to **cut.**
Machining consumes **Raw Metal Bar.**
Machining yields **pre-tested Part.**
Testing changes **Part** from **pre-tested** to **tested.**

Figure 11 — State-specified consumption and results links

NOTE 2 If an active affecting process stops prematurely, i.e. it does not complete, the state of any affectee remains indeterminate unless exception handling resolves the object to one of its permissible states.

9.3.3.3 Input-specified effect link

An input-specified effect link shall be a pair of effect links, where the input source link connects to an affecting process from an input source state of the affectee, and the output destination link connects from the same process to the same affectee without specifying a particular state. The output destination state of the object shall be its default state or, if the object does not have a default state. then the state probability distribution of the object shall determine the output destination state of that object (see 12.7).

Existence of the affectee at the input source state is a precondition, or part of the precondition, for affecting process activation. Existence of the affectee at any one of its states shall be a postcondition, or part of the postcondition, upon affecting process completion.

Graphically, a pair of arrows consisting of an arrow with a closed arrowhead from the input source state of the affectee to the affecting process, the input link, and a similar arrow from that process to the affectee but not to any one of its states shall denote the input-specified effect link.

The syntax of an input-specified effect link OPL sentence shall be: **Process** changes **Object** from **input-state.**

9.3.3.4 Output-specified effect link

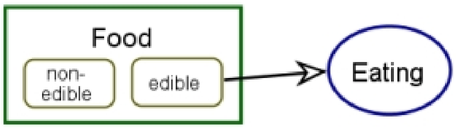
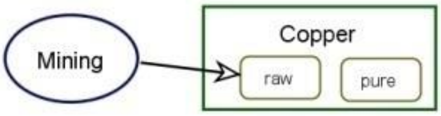
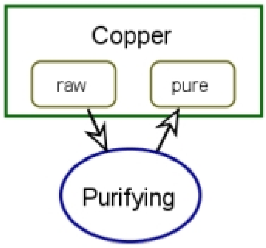
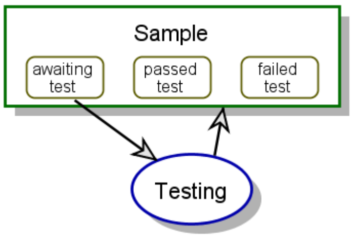
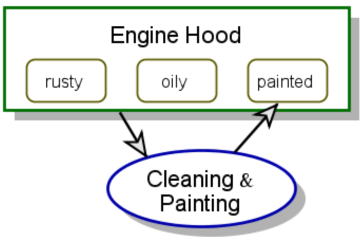
An output-specified effect link shall be a pair of effect links, where the input source link connects to an affecting process from an affectee without specifying a particular state, and the output destination link connects from the same process to an output destination state of the same affectee. Existence of the affectee shall be a precondition, or part of a precondition, for affecting process activation. Existence of the affectee at the output destination state shall be a postcondition, or part of the postcondition, upon affecting process completion.

Graphically, a pair of arrows consisting of an arrow with a closed arrowhead from the affectee without specifying a particular state, the input link, and a similar arrow from that process to an output destination state of that affectee, the output link, shall denote the output-specified effect link.

The syntax of an input-specified effect link OPL sentence shall be: **Process** changes **Object** to **output-state.**

9.3.4 State-specified transforming links summary

Table 3 — State-specified transforming links summary

Name	Semantics	Sample OPD & OPL	Source	Destination
State-specified consumption link	The process consumes the object if and only if the object is in the specified state.	 **Eating** consumes **edible Food.**	consumee state	process
State-specified result link	The process generates the object in the specified state.	 **Mining** yields **raw Copper.**	process	resultee state
Input-output-specified effect link pair (consisting of one state-specified *input link* and one state-specified *output link*)	The process changes the object from a specified input state via the *input link* to a specified output state via the *output link*.	 **Purifying** changes **Copper** from **raw** to **pure.**	affectee source state	affecting process
			affecting process	affectee destination state
Input-specified effect link pair (consisting of one state-specified *input link* and one state-unspecified *output link*)	The process changes the object from a specified input state to any output state.	 **Testing** changes **Sample** from **awaiting test.**	affectee source state	affecting process
			affecting process	affectee
Output-specified effect link pair (consisting of one state-unspecified *input link* and one state-specified *output link*)	The process changes the object from any input state to a specified output state.	 **Cleaning & Painting** changes **Engine Hood** to **painted.**	affectee	affecting process
			affecting process	affectee destination state

9.4 State-specified enabling links

9.4.1 State-specified agent link

A state-specified agent link shall be an agent link from a specified state of the agent to a process. The agent in the specified state shall be necessary for process activation and performance.

Graphically, a line with a filled circle resembling a black lollipop at the terminal end extending from the specified state of the agent object to the process it enables shall denote a state-specified agent link.

The syntax of a state-specified agent link OPL sentence shall be: **Specified-state Agent** handles **Processing**.

NOTE State name labels do not appear with beginning capital letters except when they appear at the beginning of an OPL sentence.

EXAMPLE A **Pilot** must be **sober** in order to qualify as an agent for the **Flying** process of an **Airplane**. In OPL: **Sober Pilot** handles **Flying.**

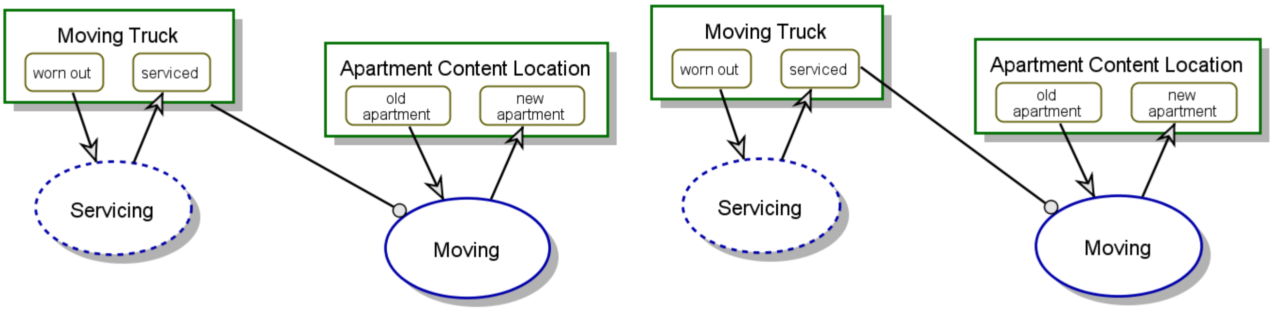
9.4.2 State-specified instrument link

A state-specified instrument link shall be an instrument link from a specified state of the instrument to a process. The instrument in the specified state shall be necessary for process activation and performance.

Graphically, a line with an empty circle resembling a white lollipop at the terminal end extending from the specified state of the instrument object to the process it enables shall denote a state-specified instrument link.

The syntax of a state-specified instrument link OPL sentence shall be: **Processing** requires **specified-state Instrument.**

EXAMPLE The OPD in Figure 12 depicts the difference between basic and state-specified instrument links. On the left, the object **Moving Truck** is the instrument for **Moving**, meaning that the state of this object does not matter, while on the right, the qualifying state **serviced** of **Moving Truck** is an instrument of **Moving**, meaning that if and only if **Moving Truck** is **serviced** may **Moving** take place.

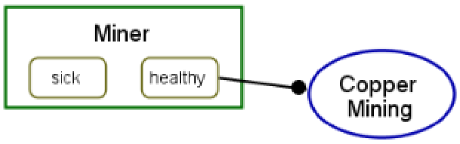
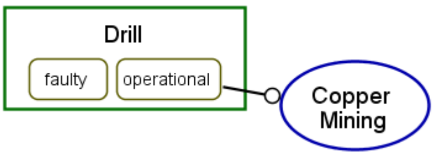


Moving Truck is **physical**. **Moving Truck** can be **worn out** or **serviced**. **Servicing** is **environmental** and **physical**. **Servicing** changes **Moving Truck** from **worn out** to **serviced**. **Apartment Content Location** is **physical**. **Apartment Content Location** can be **old apartment** or **new apartment.** **Moving** is **physical**. **Moving** requires **Moving Truck**. **Moving** changes **Apartment Content Location** from **old apartment** to **new apartment**.	**Moving Truck** is **physical**. **Moving Truck** can be **worn out** or **serviced**. **Servicing** is **environmental** and **physical**. **Servicing** changes **Moving Truck** from **worn out** to **serviced**. **Apartment Content Location** is **physical**. **Apartment Content Location** can be **old apartment** or **new apartment**. **Moving** is **physical**. **Moving** requires **serviced Moving Truck**. **Moving** changes **Apartment Content Location** from **old apartment** to **new apartment**.

Figure 12 — Instrument link on left vs. state-specified instrument link on right

9.4.3 State-specified enabling links summary

Table 4 — State specified enabling links summary

Name	Semantics	Sample OPD & OPL	Source	Destination
State-specified agent link	The human agent enables the process provided she is at the specified state.	 **Healthy Miner** handles **Copper Mining.**	agent state	enabled process
State-specified instrument link	The process requires the instrument at the specified state.	 **Copper Mining** requires **operational Drill.**	instrument state	enabled process

9.5 Control links

9.5.1 Kinds of control links

As part of the Event-Condition-Action paradigm (see 8.2.1) underlying OPM's operational semantics, an event link, a condition link, and an exception link shall express an event, a condition, and a time exception respectively. These three link kinds shall be control links. Control links shall occur either between an object and a process or between two processes.

An event link shall specify a source event and a destination process to activate upon event occurrence. The event occurrence causes an evaluation of the process' precondition for satisfaction.

Satisfying the precondition allows process performance to proceed and the process becomes active. If the process precondition is not satisfied, then process performance shall not occur. Regardless of whether the evaluation is successful or not, the event shall be lost.

If the process precondition is not satisfied, process activation shall not occur until another event activates the process. Control links determine if the process waits for another activating event or if the flow of execution control bypasses the process.

NOTE 1 Subsequent events may come from other sources to initiate precondition evaluation.

A condition link shall be a procedural link between a source object or object state and a destination process. A condition link shall provide a bypass mechanism, which enables system execution control to skip, or bypass, the destination process if its precondition satisfaction evaluation fails.

NOTE 2 Without the condition link bypass mechanism, the failure to satisfy the precondition constrains the process to wait for satisfaction of the precondition.

For both event links and condition links, each kind of incoming transforming link and enabling link, i.e. a link from an object or object state to a process, shall have a corresponding kind of event link and condition link.

An exception link shall be a procedural link between a process that for some reason is unable to complete successfully or takes more or less time to complete than expected, and a process that is to manage the exception situation.

NOTE 3 Exception links express only failures in time-based performance criteria. Since most exceptions result in undertime or overtime performance, exception links serve many situations.

Graphically, a control modifier appearing as an annotation next to an incoming transforming link or enabling link, i.e. a link from an object or an object state to a process, shall denote the corresponding control link. The symbol "e" annotation, signifying event, shall denote an event link and the symbol "c" annotation, signifying condition, shall denote a condition link. The control modifier annotation for an exception link is one or two short bars crossing the link near the exception managing process.

9.5.2 Event links

9.5.2.1 Transforming event links

9.5.2.1.1 Consumption event link

A consumption event link shall be an annotated consumption link between an object and a process, which an operational instance of the object initiates. Satisfaction of the process precondition and the subsequent process performance shall consume the instance of the initiating object.

Graphically, an arrow with a closed arrowhead pointing from the object to the process with the small letter "e" annotation near the arrowhead, signifying event, shall denote the consumption event link.

The syntax of a consumption event link OPL sentence shall be: **Object** initiates **Process**, which consumes **Object.**

9.5.2.1.2 Effect event link

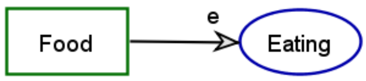
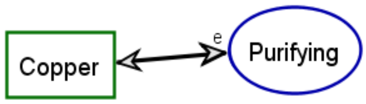
An effect event link shall be an annotated portion of an effect link from an object to a process, which an operational instance of the object initiates. Satisfaction of the process precondition and the subsequent process performance shall affect the initiating object in some manner.

Graphically, a bidirectional arrow with closed arrowheads at each end between the object and the process with a small letter "e" annotation near the process end of the arrow, signifying event, shall denote the effect event link.

The syntax of an effect event link OPL sentence shall be: **Object** initiates **Process**, which affects **Object.**

9.5.2.1.3 Transforming event links summary

Table 5 —Transforming event link summary

Name	Semantics	Sample OPD & OPL	Source	Destination
Consumption event link	The object initiates the process, which, if performed, consumes the object.	 **Food** initiates **Eating**, which consumes **Food.**	initiating consumee	initiated process, which consumes the initiating consumee
Effect event link	The object initiates the process, which, if performed, affects the object.	 **Copper** initiates **Purifying,** which affects **Copper.**	initiating affectee	initiated process, which affects the initiating affectee
			NOTE The event link is the link from the object to the process; the link from the process to the object is not an event link.	

9.5.2.2 Enabling event links

9.5.2.2.1 Agent event link

An agent event link shall be an annotated enabling link from an agent object to the process that it initiates and enables.

Graphically, a line with a filled circle resembling a black lollipop at the terminal end extending from an agent object to the process it initiates and enables with a small letter "e" annotation near the process end, signifying event, shall denote an agent event link.

The syntax of an agent event link OPL sentence shall be: **Agent** initiates and handles **Process.**

9.5.2.2.2 Instrument event link

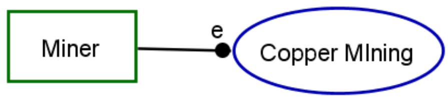
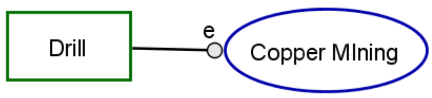
An instrument event link shall be an annotated enabling link from an instrument object to the process that it initiates and enables.

Graphically, a line with an empty circle resembling white lollipop at the terminal end extending from the instrument object to the process it initiates and enables with a small letter "e" annotation near the process end, signifying event, shall denote an instrument event link.

The syntax of an instrument event link OPL sentence shall be: **Instrument** initiates **Process,** which requires **Instrument.**

9.5.2.2.3 Enabling event link summary

Table 6 —Enabling event link summary

Name	Semantics	Sample OPD & OPL	Source	Destination
Agent event link	The agent—a human—both initiates and enables the process. The agent must exist throughout the process duration.	 **Miner** initiates and handles **Copper Mining.**	initiating agent	initiated process
Instrument event link	The object initiates the process as an instrument, so it does not change, but it must exist throughout the process duration.	 **Drill** initiates **Copper Mining,** which requires **Drill.**	initiating instrument	initiated process

9.5.2.3 State-specified transforming event links

9.5.2.3.1 State-specified consumption event link

A state-specified consumption event link shall be an annotated consumption link from a specified state of an object to a process, which an operational instance of the object initiates. Satisfaction of the process precondition, including the initiating object at the specified state, and the subsequent process performance shall consume the initiating object.

Graphically, an arrow with a closed arrowhead pointing from the specified state of the object to the process with the small letter "e" annotation near the arrowhead, signifying event, shall denote the state-specified consumption event link.

The syntax of a state-specified consumption event link OPL sentence shall be: **Specified-state Object** initiates **Process,** which consumes **Object.**

9.5.2.3.2 Input-output-specified effect event link

An input-output-specified effect event link shall be an annotated input-output-specified effect link that initiates the affecting process when an operational instance of the object enters the specified input source state.

Graphically, the input-output-specified effect link with a small letter "e" annotation near the arrowhead end of the input link, signifying event, shall denote the input-output-specified effect event link.

The syntax of an input-output-specified effect event link OPL sentence shall be: **Input-state Object** initiates **Process,** which changes **Object** from **input-state** to **output-state.**

9.5.2.3.3 Input-specified effect event link

An input-specified effect event link shall be an annotated input-specified effect link that initiates the affecting process when an operational instance of the object enters the specified input source state.

Graphically, the input-specified effect link with a small letter "e" annotation at the arrowhead end of the input link, signifying event, shall denote the input-specified effect event link.

The syntax of an input-specified effect event link OPL sentence shall be: **Input-state Object** initiates **Process,** which changes **Object** from **input-state.**

9.5.2.3.4 Output-specified effect event link

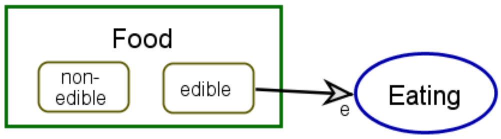
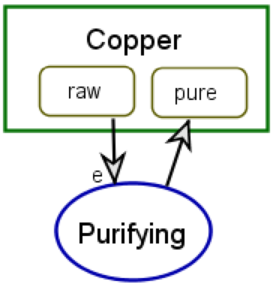
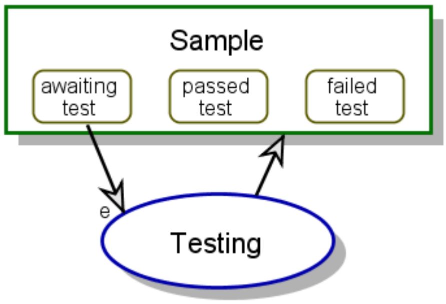
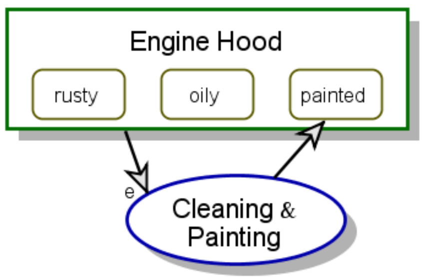
An output-specified effect event link shall be an annotated output-specified effect link that initiates the affecting process when an operational instance of the object comes into existence.

Graphically, the output-specified effect link with a small letter "e" annotation at the arrowhead end of the input link, signifying event, shall denote the output-specified effect event link.

The syntax of an output-specified effect event link OPL sentence shall be: **Object** in any state initiates **Process,** which changes **Object** to **destination-state.**

9.5.2.3.5 State-specified transforming event link summary

Table 7 — State-specified transforming event link summary

Name	Semantics	Sample OPD & OPL	Source	Destination
State-specified consumption event link	The object in the specified state both initiates the process and is consumed by it.	 **Edible Food** initiates **Eating**, which consumes **Food.**	consumee state	initiated process
Input-output specified event link pair	The object in the specified state both initiates the process and is transformed by it to the output state.	 **Raw Copper** initiates **Purifying**, which changes **Copper** from **raw** to **pure.**	affectee source state	initiates process
			initiates process	affectee destination state
Input-specified effect link pair	The object in the specified state both initiates the process and is transformed by it to any one of its states.	 **Awaiting test Sample** initiates **Testing,** which changes **Sample** from **awaiting test.**	affectee source state	initiated process
			initiates process	affectee
Output-specified event link pair	The object (in any one of its states) both initiates the process and is transformed by it to the output state.	 **Engine Hood** initiates **Cleaning & Painting,** which changes **Engine Hood** to **painted.**	affectee	initiates process
			initiates process	affectee destination state

9.5.2.4 State-specified enabling event links

9.5.2.4.1 State-specified agent event link

A state-specified agent event link shall be an annotated state-specified agent link that initiates the process when an operational instance of the agent enters the specified state.

Graphically, the state-specified agent link with a small letter "e" annotation near the process end of the link, signifying event, shall denote the state-specified agent event link.

The syntax of a state-specified agent event link OPL sentence shall be: **Specified-state Agent** initiates and handles **Processing**.

9.5.2.4.2 State-specified instrument event link

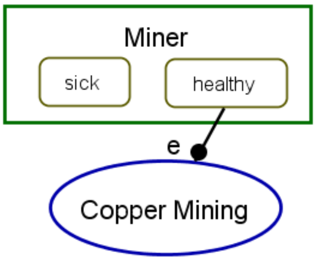
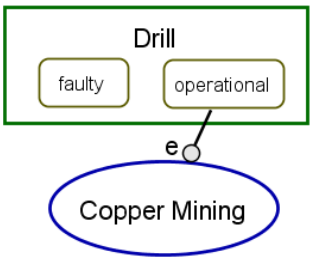
A state-specified instrument event link shall be an annotated state-specified instrument link that initiates the process when an operational instance of the instrument enters the specified state.

Graphically, the state-specified instrument link with a small letter "e" annotation near the process end of the link, signifying event, shall denote the state-specified instrument event link.

The syntax of a state-specified instrument event link OPL sentence shall be: **Specified-state Instrument** initiates **Processing**, which requires **specified-state Instrument."**

9.5.2.4.3 State-specified enabling event link summary

Table 8 — State-specified enabling event link summary

Name	Semantics	Sample OPD & OPL	Source	Destination
State-specified agent event link	The human agent in the specified state both initiates the process and acts as its agent. The agent must be at the specified state throughout the process duration.	 **Healthy Miner** initiates and handles **Copper Mining.**	agent state	initiated process
State-specified instrument event link	The object at the specified state both initiates the process and is instrument for its performance. The instrument must be at the specified state throughout the process duration.	 **Operational Drill** initiates **Copper Mining,** which requires **operational Drill.**	instrument state	initiated process

9.5.2.5 Invocation links

9.5.2.5.1 Process invocation and invocation link

Process invocation shall be an event by which a process initiates a process. An invocation link shall be a link from a source process to the destination process that it invokes (initiates), signifying that when the source process completes, it immediately initiates the destination process at the other end of the invocation link.

NOTE 1 A normal or expected flow of execution control does not invoke a new process if the prior process does not complete successfully. It is up to the modeller to take care of any process that aborts.

NOTE 2 Since an OPM process performs a transformation, the invocation link semantically implies the creation of an interim object by the invoking source process that the subsequent invoked destination process immediately consumes. In an OPM model, an invocation link may replace a transient, short-lived physical or informatical object (such as **Record ID** in a query), that a source process creates to initiate the destination process, which immediately consumes the transient object.

Graphically, a lightening symbol jagged line from the invoking source process to the invoked destination process ending with a closed arrowhead at the invoked process shall denote an invocation link.

The syntax of an invocation link OPL sentence shall be: **Invoking-process** invokes **invoked-process.**

9.5.2.5.2 Self-invocation link

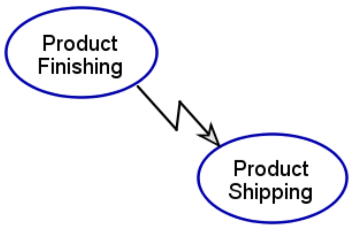
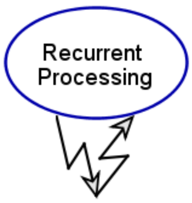
Self-invocation shall be invocation of a process by itself, such that upon process completion, the process immediately invokes itself. The self-invocation link shall specify self-invocation.

Graphically, a pair of invocation links, originating at the process and joining head to tail before ending back at the original process shall denote the self-invocation link.

The syntax of a self-invocation link OPL sentence shall be: **Invoking-process** invokes itself**.**

9.5.2.5.3 Invocation link summary

Table 9 — Invocation link summary

Name	Semantics	Sample OPD & OPL	Source	Destination
Invocation link	As soon as the invoking process ends, it invokes the process pointed to by the invocation link.	 **Product Finishing** invokes **Product Shipping.**	Initiating process	Another initiated process
Self-invocation link	Upon process completion, it immediately invokes itself.	 **Recurrent Processing** invokes itself**.**	Initiating process	The same process

9.5.3 Condition links

9.5.3.1 Basic Condition transforming links

9.5.3.1.1 Condition consumption link

A condition consumption link shall be an annotated consumption link from a consumee to a process. If a consumee operational instance exists when an event initiates the process, then the presence of that consumee operational instance satisfies the process precondition with respect to that object. If evaluation of the entire preprocess object set satisfies the precondition, the process starts and consumes that consumee instance. However, if a consumee operational instance does not exist when an event initiates the process, then the process precondition evaluation fails and the flow of execution control bypasses, or 'skips', the process without process performance.

Graphically, an arrow with a closed arrowhead pointing from the consumee to the process with the small letter "c" annotation near the arrowhead, signifying condition, shall denote a condition consumption link.

The syntax of the condition consumption link OPL sentence shall be: **Process** occurs if **Object** exists, in which case **Object** is consumed, otherwise **Process** is skipped**.**

An alternate syntax of the condition consumption link OPL sentence shall be: If **Object** exists then **Process** occurs and consumes **Object**, otherwise bypass **Process.**

9.5.3.1.2 Condition effect link

A condition effect link shall be an annotated effect link from an affectee to a process. If an affectee object operational instance exists when an event initiates the process, then the presence of that affectee instance satisfies the process precondition with respect to that object. If evaluation of the entire preprocess object set satisfies the precondition, the process starts and affects that affectee instance. However, if an affectee operational instance does not exist when an event initiates the process, then the process precondition evaluation fails and the flow of execution control bypasses, or 'skips' the process without process performance.

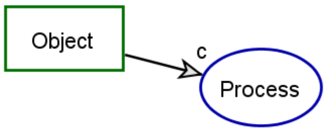
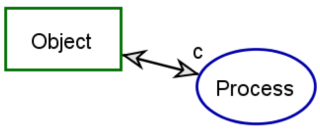
Graphically, a bidirectional arrow with two closed arrowheads, one pointing in each direction between the affectee and the affecting process, with the small letter "c" annotation near the process end of the arrow, signifying condition, shall denote a condition effect link.

The syntax of the condition effect link OPL sentence shall be: **Process** occurs if **Object** exists, in which case **Process** affects **Object**, otherwise **Process** is skipped**.**

An alternate syntax of the condition effect link OPL sentence shall be: If **Object** exists then **Process** occurs and affects **Object**, otherwise bypass **Process.**

9.5.3.1.3 Condition transforming link summary

Table 10 —Condition transforming link summary

Name	Semantics	Sample OPD & OPL	Source	Destination
Condition consumption link	If an object operational instance exists and the rest of the process precondition is satisfied, then the process performs and consumes the object instance, otherwise execution control advances to initiate the next process.	 **Process** occurs if **Object** exists**,** in which case **Process** consumes **Object,** otherwise **Process** is skipped**.**	Conditioning object	Conditioned process
Condition effect link	If an object operational instance exists and the rest of the process precondition is satisfied, then the process performs and affects the object instance, otherwise execution control advances to initiate the next process.	 **Process** occurs if **Object** exists**,** in which case **Process** affects **Object,** otherwise **Process** is skipped**.**	Conditioning object	Conditioned process

9.5.3.2 Basic condition enabling links

9.5.3.2.1 Condition agent link

A condition agent link shall be an annotated agent link from an agent to a process. If an agent operational instance exists when an event initiates the process, then the presence of that agent instance satisfies the process precondition with respect to that object. If evaluation of the entire preprocess object set satisfies the precondition, the process starts and that agent handles its performance. However, if an agent operational instance does not exist when an event initiates the process, then the process precondition evaluation fails and the flow of execution control bypasses, or 'skips' the process without process performance.

Graphically, a line with a filled circle resembling a black lollipop at the terminal end extending from an agent object to the process it enables, with the small letter "c" annotation near the process end, signifying condition, shall denote a condition agent link.

The syntax of the condition agent link OPL sentence shall be: **Agent** handles **Process** if **Agent** exists**,** else **Process** is skipped**.**

An alternate syntax for the condition agent link OPL sentence shall be: If **Agent** exists then **Agent** handles **Process**, otherwise bypass **Process**.

9.5.3.2.2 Condition instrument link

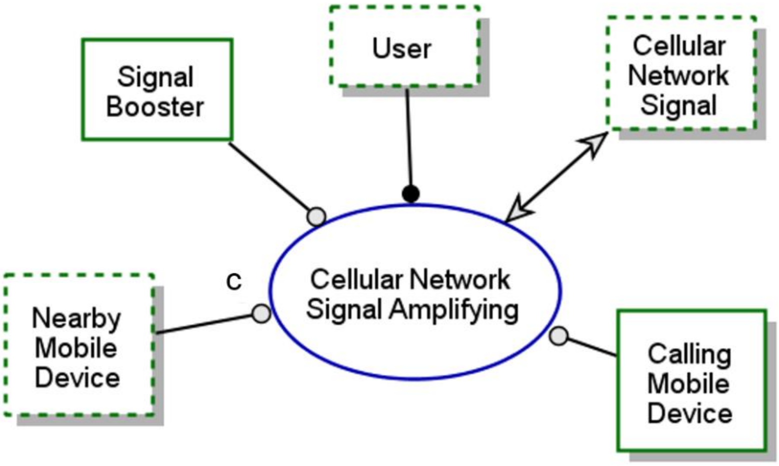
A condition instrument link shall be an annotated instrument link from an instrument to a process. If an instrument operational instance exists when an event initiates the process, then the presence of that instrument instance satisfies the process precondition with respect to that object. If evaluation of the entire preprocess object set satisfies the precondition, the process starts. However, if an instrument operational instance does not exist when an event initiates the process, then the process precondition evaluation fails and the flow of execution control bypasses, or 'skips' the process without process performance.

Graphically, a line with an empty circle resembling a white lollipop at the terminal end, extending from an instrument object to the process it enables, with the small letter "c" annotation near the process end, signifying condition, shall denote a condition instrument link.

The syntax of the condition instrument link OPL sentence shall be: **Process** occurs if **Instrument** exists**,** else **Process** is skipped**.**

An Alternate syntax for the condition instrument link OPL sentence shall be: If **Instrument** exists then Process occurs**,** otherwise bypass **Process.**

EXAMPLE Figure 13 is an OPD with a condition instrument link from **Nearby Mobile Device** to **Cellular Network Signal Amplifying**, which occurs only if an environmental object **Nearby Mobile Device** exists and is otherwise skipped, as there is no point in amplifying if no device is nearby.

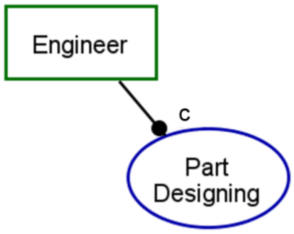
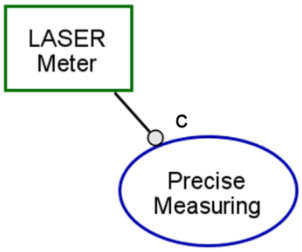


Cellular Network Signal Amplifying occurs if **Nearby Mobile Device** exists**,** otherwise **Cellular Network Signal Amplifying** is skipped**.**

Figure 13 — Condition instrument link (with partial OPL)

9.5.3.2.3 Basic condition enabling link summary

Table 11 — Condition enabling link summary

Name	Semantics	Sample OPD & OPL	Source	Destination
Agent condition link	The agent enables the process if the agent is present, otherwise the process is skipped.	 **Engineer** handles **Part Designing** if **Engineer** is present**,** otherwise **Part Designing** is skipped**.**	Conditioning agent	Conditioned process
Instrument condition link	The instrument enables the process if it exists, otherwise the process is skipped.	 **Precise Measuring** occurs if **LASER Meter** exists**,** otherwise **Precise Measuring** is skipped**.**	Conditioning instrument	Conditioned process

9.5.3.3 Condition state-specified transforming links

9.5.3.3.1 Condition state-specified consumption link

A condition state-specified consumption link shall be an annotated condition consumption link from a specified state of a consumee to a process. If an operational instance of the consumee at the specified state exists when an event initiates the process, then the presence of that consumee instance satisfies the process precondition with respect to that object. If evaluation of the entire preprocess object set satisfies the precondition, the process starts and consumes that consumee instance. However, if an operational instance of a consumee in the specified state does not exist when an event initiates the process, then the process precondition evaluation fails and the flow of execution control bypasses, or 'skips', the process without process performance.

Graphically, an arrow with a closed arrowhead pointing from the specified state of the consumee to the process with the small letter "c" annotation near the arrowhead, signifying condition, shall denote a condition state-specified consumption link.

The syntax of the condition state-specified consumption link OPL sentence shall be: **Process** occurs if **Object** is **specified-state**, in which case **Object** is consumed**,** otherwise **Process** is skipped**.**

An alternate syntax for the condition state-specified consumption link OPL sentence shall be: If **specified-state Object** exists then **Process** occurs and consumes **Object,** otherwise bypass **Process.**

9.5.3.3.2 Condition input-output-specified effect link

A condition input-output-specified effect link shall be an annotated input-output-specified effect link from a source input state to a process. If an operational instance of the affectee at the specified state exists when an event initiates the process, then the presence of that affectee instance satisfies the process precondition with respect to that object. If evaluation of the entire preprocess object set satisfies the precondition, the process starts and affects that object operational instance by changing the state of the instance from the specified input state to the specified output state. However, if an operational instance of an affectee at the specified state does not exist when an event initiates the process, then the process precondition evaluation fails and the flow of execution control bypasses, or 'skips', the process without process performance.

Graphically, the condition input-output-specified effect link with the small letter "c" annotation near the arrowhead of the input link, signifying condition, shall denote a condition input-output-specified effect link.

The syntax of the condition input-output-specified effect link OPL sentence shall be: **Process** occurs if **Object** is **input-state**, in which case **Process** changes **Object** from **input-state** to **output-state,** otherwise **Process** is skipped**.**

An alternate syntax for the condition input-output-specified effect link OPL sentence shall be: If **input-state Object** then **Process** changes **Object** from **input-state** to **output-state,** otherwise bypass **Process.**

9.5.3.3.3 Condition input-specified effect link

A condition input-specified effect link shall be an annotated input-specified effect link from a source input state to a process. If an operational instance of the affectee at the specified state exists when an event initiates the process, then the presence of that affectee instance satisfies the process precondition with respect to that object. If evaluation of the entire preprocess object set satisfies the precondition, the process starts and affects that object instance by changing the state of the instance from the specified input state to a destination state. The destination state shall be either its default state or, if the object does not have a default state, the state probability distribution of the object shall determine the output destination state of that object (see 12.7). However, if an operational instance of an affectee at the specified state does not exist when an event initiates the process, then the process precondition evaluation fails and the flow of execution control bypasses, or 'skips', the process without process performance.

Graphically, the condition input-specified effect link with the small letter "c" annotation near the arrowhead of the input link, signifying condition, shall denote the condition input-specified effect link.

The syntax of a condition input-specified effect link OPL sentence shall be: **Process** occurs if **Object** is **input-state**, in which case **Process** changes **Object** from **input-state,** otherwise **Process** is skipped**.**

An alternate syntax for a condition input-specified effect link OPL sentence shall be: if **input-state Object** then **Process** changes **Object** from **input-state,** otherwise bypass **Process.**

9.5.3.3.4 Condition output-specified effect link

A condition output-specified effect link shall be an annotated output-specified effect link from a source object to a process. If an operational instance of the affectee exists when an event initiates the process, then the presence of that affectee instance satisfies the process precondition with respect to that object. If evaluation of the entire preprocess object set satisfies the precondition, the process starts and affects that object instance by changing the state of the instance to the specified output-state. However, if an operational instance of an affectee does not exist when an event initiates the process, then the process precondition evaluation fails and the flow of execution control bypasses, or 'skips', the process without process performance.

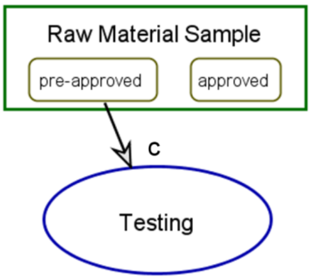
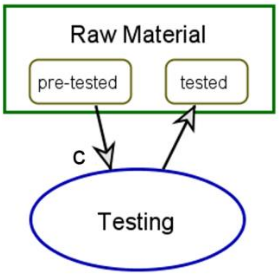
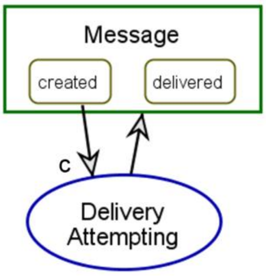
Graphically, the condition output-specified effect link with the small letter "c" annotation near the arrowhead of the input link, signifying condition, shall denote a condition output-specified effect link.

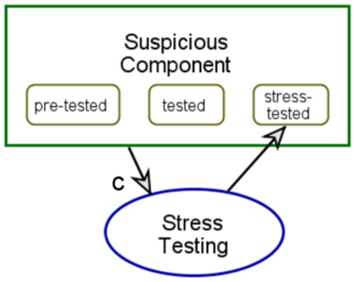
The syntax of the condition output-specified effect OPL sentence shall be: **Process** occurs if **Object** exists, in which case **Process** changes **Object** to **output-state,** otherwise **Process** is skipped**.**

An alternate syntax for the condition output-specified effect OPL sentence shall be: if **Object** exists then **Process** changes **Object** to **output-state,** otherwise bypass **Process.**

9.5.3.3.5 Condition state-specified transforming link summary

Table 12 — Condition state-specified transforming link summary

Name	Semantics	Sample OPD & OPL	Source	Destination
Condition state-specified consumption link	The process performs if the object is in the state from which the link originates, otherwise the process is skipped.	 **Testing** occurs if **Raw Material Sample** is **pre-approved,** in which case **Raw Material Sample** is consumed**,** otherwise **Testing** is skipped**.**	conditioning specified state of the object	conditioned process
Condition input-output-specified effect link	The process performs if the object is in the input state (from which the link originates) and changes the object from its input state to its output state, otherwise the process is skipped.	 **Testing** occurs if **Raw Material** is **pre-tested,** in which case **Testing** changes **Raw Material** from **pre-tested** to **tested,** otherwise **Testing** is skipped**.**	conditioning specified input state of the object	conditioned process
Condition input-specified effect link	The process performs if the object is in the input state (from which the link originates) and changes the object from its input state to any one of its states, otherwise the process is skipped.	 **Delivery Attempting** occurs if **Message** is **created**, in which case **Delivery Attempting** changes **Message** from **created,** otherwise **Delivery Attempting** is **skipped.**	conditioning specified input state of the object	conditioned process

Condition output-specified effect link	The process performs if the object is in the input state (from which the link originates) and changes the object from its input state to any one of its states, otherwise the process is skipped.	 **Stress Testing** occurs if **Suspicious Component** exists**,** in which case **Stress Testing** changes **Suspicious Componen**t to **stress-tested,** otherwise **Stress Testing** is skipped**.**	conditioning object	conditioned process

9.5.3.4 Condition state-specified enabling links

9.5.3.4.1 Condition state-specified agent link

A condition state-specified agent link shall be an annotated state-specified agent link from a specified state of an agent to a process. If an operational instance of the agent at the specified state exists when an event initiates the process, then the presence of that agent instance satisfies the process precondition with respect to that object. If evaluation of the entire preprocess object set satisfies the precondition, the process starts and that agent handles operation. However, if an operational instance of an agent in the specified state does not exist when an event initiates the process, then the process precondition evaluation fails and the flow of execution control bypasses, or 'skips', the process without process performance.

Graphically, the state-specified agent link with a small letter "c" annotation near the process end, signifying condition, shall denote a condition state-specified agent link.

The syntax of the condition state-specified agent link OPL sentence shall be: **Agent** handles **Process** if **Agent** is **specified-state,** else **Process** is skipped**.**

An alternate syntax for the condition state-specified agent link OPL sentence shall be: If **specified-state Agent** exists then **Agent** handles **Process,** otherwise bypass **Process.**

9.5.3.4.2 Condition state-specified instrument link

A condition state-specified instrument link shall be an annotated state-specified instrument link from a specified state of an instrument to a process. If an operational instance of the instrument at the specified state exists when an event initiates the process, then the presence of that instrument instance satisfies the process precondition with respect to that object. If evaluation of the entire preprocess object set satisfies the precondition, the process starts. However, if an operational instance of an instrument in the specified state does not exist when an event initiates the process, then the process precondition evaluation fails and the flow of execution control bypasses, or 'skips', the process without process performs.

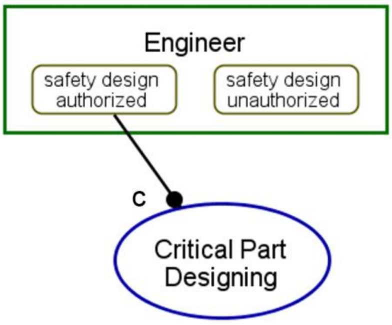
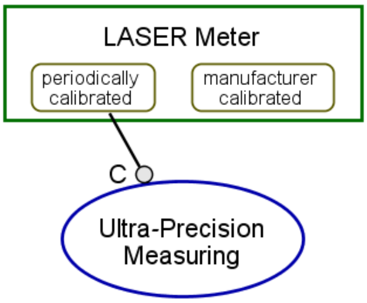
Graphically, the state-specified instrument link with a small letter "c" annotation near the process end, signifying condition, shall denote a condition state-specified instrument link.

The syntax of the condition state-specified instrument link OPL sentence shall be: "**Process** occurs if **Instrument** is **specified-state,** otherwise **Process** is skipped**.**

An alternate syntax for the condition state-specified instrument link OPL sentence shall be: If **specified-state Instrument** then **Process** occurs**,** otherwise bypass **Process.**

9.5.3.4.3 Condition state-specified enabling link summary

Table 13 — Condition state-specified enabling link summary

Name	Semantics	Sample OPD & OPL	Source	Destination
State-specified agent condition link	The agent enables the process if the agent is in the specified state, otherwise the process is skipped.	 **Engineer** handles **Critical Part Designing** if **Engineer** is **safety design authorized,** otherwise **Critical Part Designing** is skipped**.**	conditioning specified state of agent	conditioned process
State-specified instrument condition link	The instrument enables the process if it is in the specified state, otherwise the process is skipped.	 **Ultra-Precision Measuring** occurs if **LASER Meter** is **periodically calibrated,** otherwise Precise Measuring is skipped**.**	conditioning specified state of instrument	conditioned process

9.5.4 Exception links

9.5.4.1 Minimal, Expected, and Maximal Process Duration and Duration Distribution

A process may have a **Duration** attribute with a value that expresses units of time. **Duration** may specialize into **Minimal Duration**, **Expected Duration**, and **Maximal Duration.**

Minimal Duration and **Maximal Duration** should designate the minimum and maximum allowable time units for process completion. **Expected Duration** of a process should be the statistical mean of the duration of that process.

Duration may have an optional **Duration Distribution** property with a value identifying the name and parameters for a probability distribution function associated with the process duration. At run-time, the value of **Duration** is determined separately for each process instance (i.e. for each individual process occurrence) by sampling from the process **Duration Distribution.**

NOTE See Annex C for process duration and system time run-time discussion and examples.

9.5.4.2 Overtime exception link

The overtime exception link shall connect the source process with an overtime handling destination process to specify that if at runtime, performance of the source process instance exceeds its **Maximal Duration** value, then an event initiates the destination process.

Graphically, a single short bar, oblique to the line connecting the source and destination processes and next to the destination process, shall denote the overtime exception link.

Given that, **max-duration** is the value of **Maximal Duration**, and **time-unit** is an allowable time measurement unit, the syntax of the overtime exception link shall be: **Overtime Handling Destination Process** occurs if duration of **Source Process** exceeds **max-duration time-units**.

9.5.4.3 Undertime exception link

The undertime exception link shall connect the source process with an undertime handling destination process to specify that if at runtime, performance of the source process instance takes less than its **Minimal Duration** value, then an event initiates the destination process.

Graphically, two parallel short bars, oblique to the line connecting the source and destination processes and next to the destination process, shall denote the undertime exception link.

Given that, **min-duration** is the value of Minimal Duration, and **time-unit** is an allowable time measurement unit, the syntax of the undertime exception link shall be: **Undertime Handling Destination Process** occurs if duration of **Source Process** falls short of **min-duration time-units.**

NOTE Similar to the invocation link, the two time exception links are procedural links that connect two processes directly, unlike most procedural links, which connect an object and a process. There is, in fact, an interim object **Overtime Exception Message** or an **Undertime Exception Message** created by the OPM's process execution mechanism realizing the process failed to end by the maximal allotted time or ended prematurely, falling short of the minimal allotted time, respectively. Since the OPM operational mechanism creates and immediately consumes these objects, their depiction is not necessary in the model.

10 Structural links

10.1 Kinds of structural links

Structural links specify static, time-independent, long-lasting relations in the system. A structural link shall connect two or more objects or two or more processes, but not an object and a process, except in the case of an exhibition-characterization link (see 10.3.3). The two kinds of structural links shall be tagged structural links and fundamental structural links of aggregation-participation, exhibition-characterization, generalization-specialization, and classification-instantiation.

10.2 Tagged structural link

10.2.1 Unidirectional tagged structural link

A unidirectional tagged structural link shall have a user-defined semantics regarding the nature of the relation from one thing to the other thing. A meaningful tag, in the form of a textual phrase, shall express the nature of the structural relation between the connecting objects or connecting processes. The tag should convey that meaning when placed in the OPL sentence.

Graphically, an arrow with an open arrowhead and a tag annotation near the shaft shall denote a unidirectional tagged structural link.

The syntax of the unidirectional tagged structural link OPL sentence shall be: **Source-thing tag Destination-thing.**

NOTE Since the tag is a label added to the model by the modeller, in the OPL sentence the tag phrase appears in bold to distinguish it from other words implicit in the syntactic construction.

10.2.2 Unidirectional null-tagged structural link

A unidirectional null-tagged structural link shall be a unidirectional tagged structural link with no tag annotation, signifying the use of the default unidirectional tag. The default tag shall be "relates to".

The syntax of the unidirectional null-tagged structural link OPL sentence shall be: **Source-thing** relates to **Destination-thing.**

NOTE The modeller should have the option of setting the default unidirectional tag, which does not appear in bold letters, for a specific system or a set of systems.

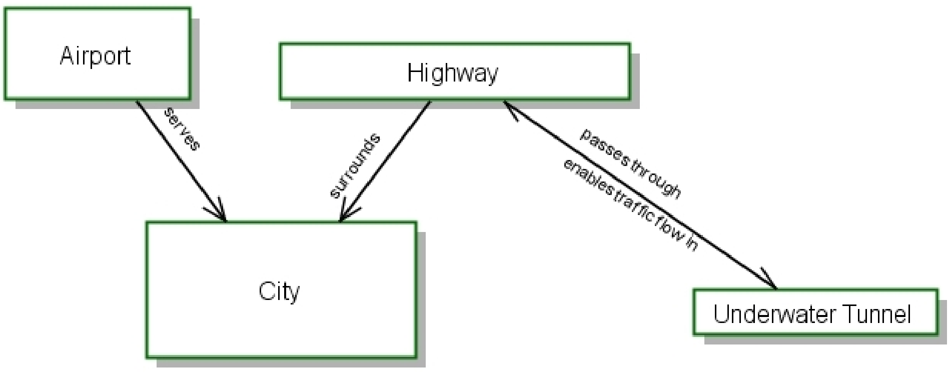
10.2.3 Bidirectional tagged structural link

Because relations between things are bidirectional, every tagged structural link has a corresponding tagged structural link in the opposite direction. When the tags in both directions are meaningful and not just the inverse of each other, they may be annotated by two tags on either side of a single bidirectional tagged structural link.

Graphically, a line with harpoon shaped arrowheads on opposite sides at both ends of the link shall denote a bidirectional tagged structural link. Each tag shall align on the side of the arrow with the harpoon edge sticking out of the arrowhead, unambiguously determining the direction in which each relation applies.

The syntax of the resulting tagged structural link shall be two separate unidirectional tagged structural link OPL sentences, one for each direction.

EXAMPLE



Airport serves City.
Highway surrounds City.
Highway passes through Underwater Tunnel.
Underwater Tunnel enables traffic flow in Highway.

Figure 14 — Two kinds of tagged structural links

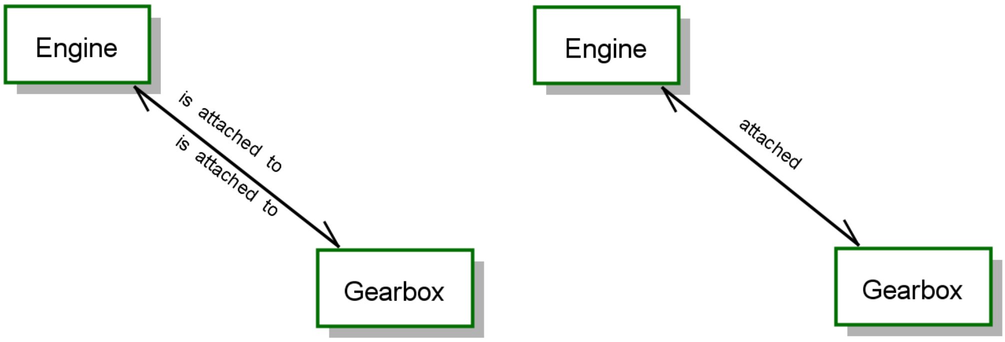
10.2.4 Reciprocal tagged structural link

A reciprocal tagged structural link shall be a bidirectional tagged structural link with only one tag or no tag. In either case, reciprocity shall indicate that the tag of a bidirectional structural link has the same semantics for each direction of the relation. When no tag appears, the default tag shall be "are related".

The syntax of the reciprocal tagged structural link with only one tag shall be: **Source-thing** and **Destination-thing** are **reciprocity-tag**.

The syntax of the reciprocal tagged structural link with no tag shall be: **Source-thing** and **Destination-thing** are related**.**

EXAMPLE In Figure 15, on the right is the reciprocal structure link equivalent to the bidirectional tagged structure link on the left, which has the same tag in each direction.



Engine is attached to Gearbox.
Gearbox is attached to Engine.

Engine and **Gearbox** are **attached.**

Figure 15 — Bidirectional (left) and its equivalent reciprocal tagged structural link (right)

NOTE As shown in Figure 15, a change in verb or noun form from that of the bidirectional tagged structural link is usually necessary to accommodate the reciprocal tagged structural link syntax.

10.3 Fundamental structural relations

10.3.1 Kinds of fundamental structural relations

The fundamental structural relations are the most prevalent structural relations among OPM things and are of particular significance for specifying and understanding systems. Each of the fundamental relations shall elaborate or refine one source thing, the refineable, into a collection of one or more destination things, the refinees.

The fundamental structural relations shall be:

- Aggregation-participation, which designates the relation between a whole and its parts;
- Exhibition-characterization, which designates the relation between an exhibitor, a thing exhibiting one or more features (attributes and/or operations), and the things that characterize the exhibitor;
- Generalization-specialization, which designates the relation between a general thing and its specializations; and
- Classification-instantiation, which designates the relation between a class of things and a refinee instance of that class.

Aggregation, exhibition, generalization, and classification shall be the refinement relation identifiers, i.e., the identifiers associated with the relation as seen from the perspective of the refineable. Participation, characterization, specialization, and instantiation shall be the corresponding complementary relation identifiers, i.e. the relation identifiers as seen from the perspective of their refinees.

With the exception of exhibition-characterization, the refinee destination things shall all have the same Perseverance value as the refineable source thing, i.e. either all are objects with static Perseverance or all are processes with dynamic Perseverance.

Folding the refines shall be the hiding of those refines of a refineable, and unfolding the refineable shall be the expressing of the refinees of that refineable (see 14.2.1.2).

Because the fundamental structural relations are bidirectional, the associated OPL paragraph could provide sentences for each direction. However, since one of these sentences is always the consequence of the other, the OPL expression of a fundamental structural relation shall be limited to one of the two possible sentences. The presentation of each kind of fundamental structural relation includes the specification of the default OPL sentence for only one of the two possible sentences. Table 14 summarizes these default sentences.

The collection of refinees modelled for some refineable in some OPD may be complete or incomplete, i.e. the graphical figure explicitly depicts, and the corresponding text explicitly expresses, only those things relevant to the OPD in which the structural link appears.

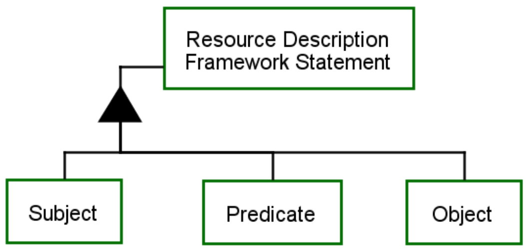
10.3.2 Aggregation-participation relation link

The fundamental structural relation aggregation-participation shall mean that a refineable, the whole, aggregates one or more refinees, the parts.

Graphically, a black solid (filled in) triangle with its apex connecting by a line to the whole and the parts connecting by lines to the opposite horizontal base shall denote the aggregation-participation relation link.

The syntax of the aggregation-participation relation link shall be: **Whole-thing** consists of **Part-thing**$_1$, **Part-thing**$_2$, …, and **Part-thing**$_n$**.**

EXAMPLE 1



Resource Description Framework Statement consists of **Subject, Predicate,** and **Object.**

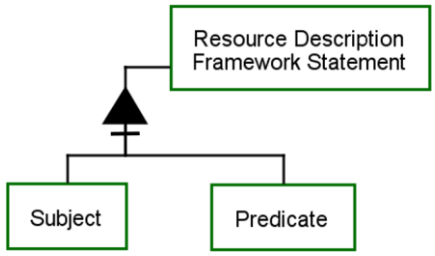
Figure 16 — Aggregation-participation relation link

When the representation of the collection of parts at the particular extent of detail is incomplete, the aggregation-participation relation link shall signify the incomplete representation with an annotation.

Graphically, a short horizontal bar crossing the vertical line below the black triangle shall denote the incomplete aggregation-participation relation link.

The syntax of the aggregation-participation relation link indicating a partial collection of parts where at least one part is missing shall be: **Whole-thing** consists **of Part-thing**$_1$, **Part-thing**$_2$,… **Part-thing**$_k$, and at least one other part**.**

EXAMPLE 2 In Figure 17, **Object** from Figure 16 is missing. The short horizontal bar crossing the vertical line below the black triangle denotes the missing thing**.**



Resource Description Framework Statement consists of **Subject, Predicate,** and at least one other part**.**

Figure 17 — Aggregation-participation relation link example with partial refinee set

EXAMPLE 3 On the left in Figure 18, the Consuming process consumes the Whole along with its Part B and Part D, while Part A and Part C remain as separate objects. On the right in Figure 18, the terse version using partial aggregation shows the Consuming process consumes the Whole and only Part B and Part D, while other parts of the Whole remain as distinct objects.

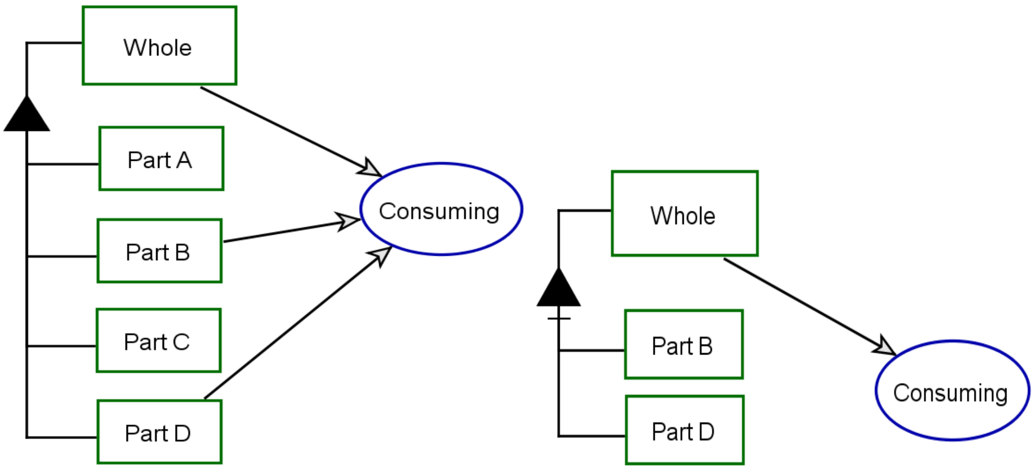


Figure 18 - Partial aggregation consumption

NOTE A tool should keep track of the set of refinees for each refineable and adjust the symbol and corresponding OPL sentences (specified below for each fundamental structural relation link) as the modeller changes the collection of refinees.

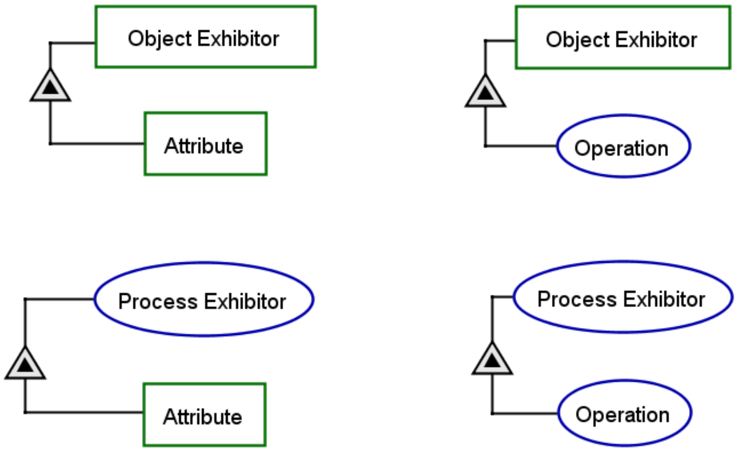
10.3.3 Exhibition-characterization link

10.3.3.1 Exhibition-characterization relation link expression

The fundamental structural relation exhibition-characterization shall mean that a refineable, the exhibitor, exhibits one or more features that characterize the exhibitor, the refinees. The features shall characterize the exhibitor.

A feature shall be a thing. An attribute shall be a feature that is an object. An operation shall be a feature that is a process. A process exhibitor and an object exhibitor shall each have at least one feature and may have both attributes, their object features, and operations, their process features.

The exhibition-characterization relation can combine the four exhibitor-feature combinations of object and process (see Figure 19).



Object Exhibitor exhibits **Attribute.** **Object Exhibitor** exhibits **Operation.**
Process Exhibitor exhibits **Attribute.** **Process Exhibitor** exhibits **Operation.**

Figure 19 — The four exhibition-characterization feature combinations

Graphically, a smaller black triangle inside a larger empty triangle with that larger triangle's apex connecting by a line to the exhibitor and the features connecting to the opposite (horizontal) base shall denote the exhibition-characterization relation link (see Figure 19).

The syntax of the exhibition-characterization relation link for an object exhibitor with a complete collection of n attributes and m operations shall be: **Object-exhibitor** exhibits **Attribute**$_1$**, Attribute**$_2$**,** … **,** and **Attribute**$_n$**,** as well as **Operation**$_1$**, Operator**$_2$**,** … **, Operator**$_m$**.**

The syntax of the exhibition-characterization relation link for a process exhibitor with a complete collection of n operation features and m attribute features shall be: **Process-exhibitor** exhibits **Operation**$_1$**, Operator**$_2$**,** … **, Operator**$_n$**,** as well as **Attribute**$_1$**, Attribute**$_2$**,** …**,** and **Attribute**$_m$**.**

NOTE 1 In the OPL for exhibition-characterization, for an object exhibitor the list of attributes precedes the list of operations, while for a process exhibitor the list of operations precedes the list of attributes.

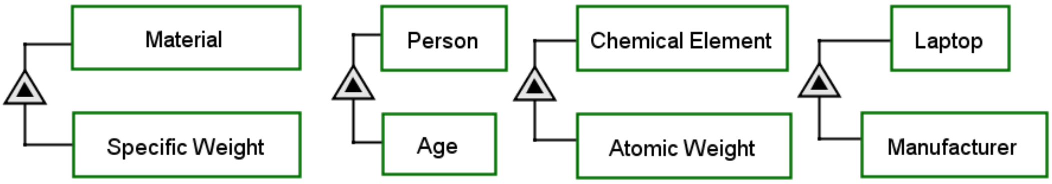
When the representation of the collection of features at the particular extent of detail is incomplete, the exhibition-characterization relation link shall signify the incomplete representation with an annotation.

Graphically, a short horizontal bar crossing the vertical line below the larger empty triangle denotes the incomplete exhibition-characterization relation link.

The syntax of the exhibition-characterization relation link for an object exhibitor with a partial collection of j attribute features and k operation features shall be: **Object-exhibitor-thing** exhibits **Attribute**$_1$**, Attribute**$_2$**,** …, **Attribute**$_j$**,** and at least one other attribute**,** as well as **Operation**$_1$**, Operator**$_2$**,** …**, Operator**$_k$**,** and at least another operation**.**

The syntax of the exhibition-characterization relation link for a process exhibitor with a partial collection of j operation features and k attribute features shall be: **Process-exhibitor** exhibits **Operation**$_1$**, Operator**$_2$**,** … **, Operator**$_j$**,** and at least another operation**,** as well as **Attribute**$_1$**, Attribute**$_2$**,** …**, Attribute**$_k$**,** and at least one other attribute**.**

EXAMPLE Figure 20 through Figure 23 show the four exhibitor-feature combinations of object and process.



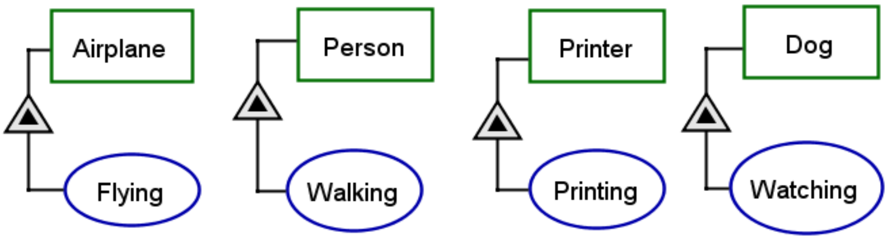
Material exhibits **Specific Weight.**

Person exhibits **Age.**

Chemical Element exhibits **Atomic Weight.**

Laptop exhibits **Manufacturer.**

Figure 20 — Object attribute examples



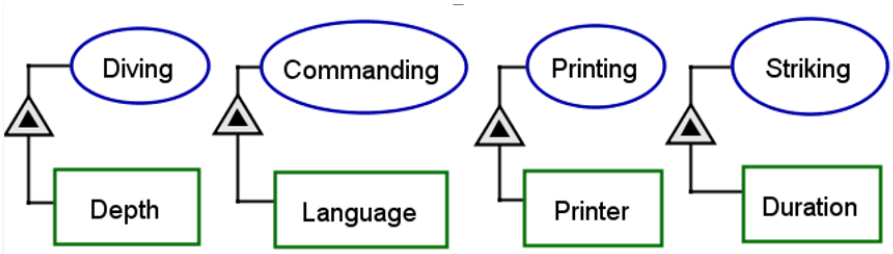
Airplane exhibits **Flying.**

Person exhibits **Walking.**

Printer exhibits **Printing.**

Dog exhibits **Watching.**

Figure 21 — Object exhibitor with operation examples



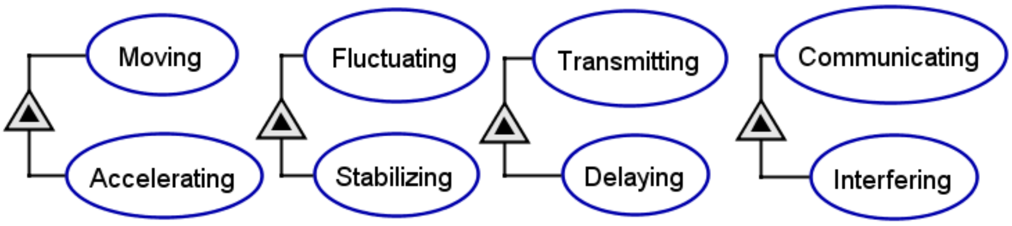
Diving exhibits **Depth.**

Commanding exhibits **Language.**

Printing exhibits **Printer.**

Striking exhibits **Duration.**

Figure 22 — Process exhibitor with attribute examples



Moving exhibits **Accelerating.**

Fluctuating exhibits **Stabilizing.**

Transmitting exhibits **Delaying.**

Communicating exhibits **Interfering.**

Figure 23 — Process exhibitor with operation examples

NOTE 2 A tool should keep track of the set of refinees for each refineable and adjust the symbol and corresponding OPL sentences (specified below for each fundamental structural relation link) as the modeller changes the collection of refinees.

10.3.3.2 Attribute state and exhibitor features

10.3.3.2.1 Attribute state as value

An attribute state, i.e. a state of the object that is the refinee attribute, shall be a value for that attribute. The static, conceptual model, shall identify all possible values for the attribute. Some may be ranges of values, while the dynamic, operational instance model shall indicate the actual attribute value at the time of the attribute's inspection (see EXAMPLE 1 and EXAMPLE 2 in 10.3.5.1.).

10.3.3.2.2 Expressing exhibitor-feature relation

When expressing features or values for an attribute, the model shall identify the exhibitor of that feature or value. To specify the exhibitor of the feature, the relation "of" shall occur in OPL sentences between the feature and its exhibitor.

The syntax for an OPL sentence identifying the exhibitor-feature relation shall be: **Feature** of **Exhibitor …**

EXAMPLE 1 In Figure 27, the OPL sentence indicating the ownership of the attribute **Specific Weight** by its **Metal Powder Mixture** exhibitor is: **Specific Weight** in **gr/cm3** of **Metal Powder Mixture** ranges from **7.545 to 7.537.**

EXAMPLE 2 In Figure 25, the OPL sentence indicating the ownership of the attribute **Travelling Medium** by its **Ship** exhibitor is: **Travelling Medium** of **Ship** is **water surface.**

10.3.4 Generalization-specialization and Inheritance

10.3.4.1 Generalization-specialization relation link

The fundamental structural relation generalization-specialization shall mean that a refineable, the general, generalizes two or more refinees, which are specializations of the general. The generalization-specialization relation binds one or more specializations with the same Perseverance as the general, such that both the general and all its specializations are objects or the general and all its specializations are processes.

Graphically, an empty triangle with its apex connecting by a line to the general and the specializations connecting by lines to the opposite base shall denote the generalization-specialization relation link (see Figure 24.

For a complete collection of n specializations of a general that is an object, the syntax of the generalization-specialization relation link OPL sentence shall be: **Specialization-object$_1$, Specialization-object$_2$, …,** and **Specialization-object$_n$** are **General-object.**

For a complete collection of n specializations of a general that is a process, the syntax of the generalization-specialization relation link OPL sentence shall be: **Specialization-process$_1$, Specialization-process$_2$, …,** and **Specialization-process$_n$** are **General-process.**

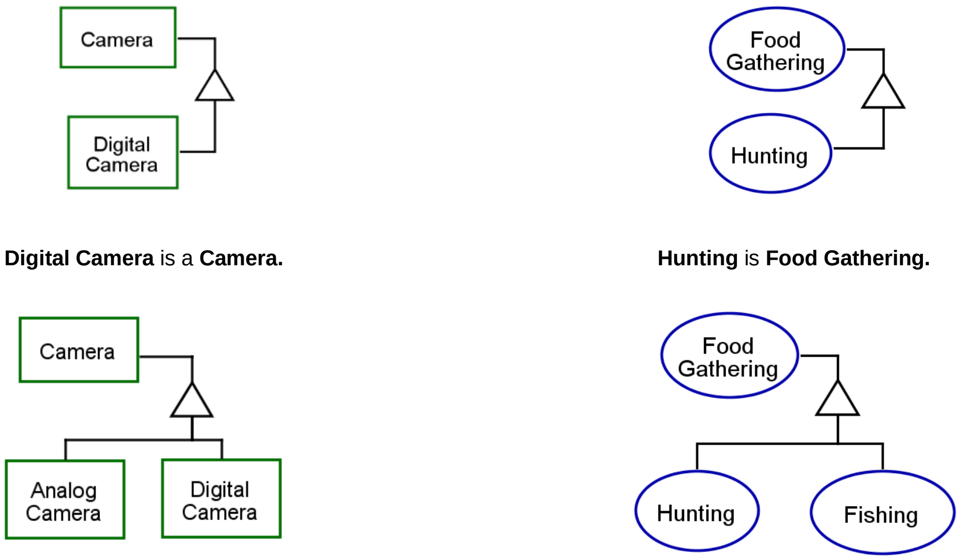
When the representation of the collection of specializations at the particular extent of detail is incomplete, the generalization-specialization relation link shall signify the incomplete representation with an annotation.

Graphically, a short horizontal bar crossing the vertical line below the empty triangle shall denote the incomplete generalization-specialization relation link.

For an incomplete set of k specializations of a general that is an object, the syntax of the generalization-specialization relation link OPL sentence shall be: **Specialization-object$_1$, Specialization-object$_2$, …, Specialization-object$_k$,** and other specializations are **General-object.**

For an incomplete set of k specializations of a general that is a process, the syntax of the generalization-specialization relation link OPL sentence shall be: **Specialization-process$_1$, Specialization-process$_2$, …, Specialization-process$_k$,** and other specializations are **General-process.**

EXAMPLE



Digital Camera is a **Camera.**

Hunting is **Food Gathering.**

Analog Camera and **Digital Camera** are **Cameras.**

Hunting and **Fishing** are **Food Gathering.**

Figure 24 — Single and plural specializations of objects and processes

NOTE A tool should keep track of the set of refinees for each refineable and adjust the symbol and corresponding OPL sentences for each fundamental structural relation link as the modeller changes the collection of refinees.

10.3.4.2 Inheritance through specialization

Inheritance shall be assignment of OPM elements, things and links, of a general to its specializations.

A specialization thing shall inherit from the general thing through the generalization-specialization link each of the following four kinds of inheritable elements that exist:

- all the parts of a general from its aggregation-participation link;
- all the features of the general from its exhibition-characterization link;
- all the tagged structural links to which the general connects; and
- all the procedural links to which the general connects.

OPM shall provide the opportunity for multiple inheritances by allowing a thing to inherit from more than one general thing each of the refines - the four inheritable elements (participants, features, tagged structural links, and procedural links) that exist for that general thing.

The modeller may override any of the participants of the general thing, which are by default inherited by the specialization, by specifying for any participant inherited from a general, a specialization of that participant with a different name and a different set of states.

NOTE When a generalization-specialization relation link exists, at runtime the specialized thing instance does not exist in the absence of the more general thing instance that it specializes and from which it inherits each of the four kinds of inheritable elements.

To create a general from one or more candidate specializations, the inheritable elements common to each of the candidates shall be migrated to a generalization thing. The manipulation of inheritable elements shall be as follows:

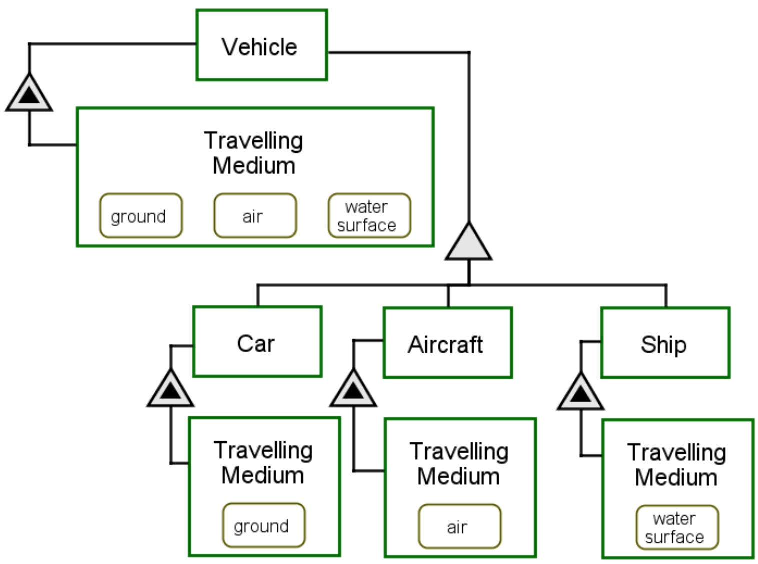
- Combine all of the common features and common participants of the specializations into one newly created general;
- Connect the new general using the generalization-specialization relation link to the specializations;
- Remove from the specializations all of the common features and common participants, which the specializations now inherit from the new general; and
- Migrate any common tagged structural links and any common procedural link edge that connects to all the specializations from the specializations to the general.

10.3.4.3 Specialization restriction through discriminating attribute

The possible values of an attribute inherited from a general may restrict the permissible value of a specialization. An inherited attribute with different values that constrain distinct values for corresponding specialization characteristics shall be a discriminating attribute.

NOTE A specialization inherits the features, and possible attribute values, of its generalization. Elaborating the general through refinement allows for a more precise valuation of inherited attributes, including specification of attribute value appropriate for the specialization's characterization through the exhibition-characterization refinement that it inherits (see also 10.4.1)

EXAMPLE 1 Figure 25 shows an OPD in which **Vehicle** exhibits the attribute **Travelling Medium** with values **ground**, **air**, and **water surface**. **Travelling Medium** is the discriminating attribute of **Vehicle**, because it constrains the specializations of **Vehicle** to values of its **Travelling Medium. Vehicle** has specializations **Car**, **Aircraft**, and **Ship**, with the corresponding **Travelling Medium** values **ground**, **air**, and **water surface**.



Vehicle exhibits **Travelling Medium.**
Travelling Medium of **Vehicle** can be **ground**, **air**, and **water surface**.
Car, **Aircraft**, and **Ship** are **Vehicles.**
Travelling Medium of **Car** is **ground**.
Travelling Medium of **Aircraft** is **air.**
Travelling Medium of **Ship** is **water surface.**

Figure 25 — The discriminating attribute Travelling Medium and its specializations

A general may have more than one discriminating attribute. The maximum number of specializations with more than one discriminating attribute shall be the Cartesian product of the number of possible values for each discriminating attribute, where some combination of attribute values may be invalid.

EXAMPLE 2 Extending the content of Figure 25, another attribute of **Vehicle** might be **Purpose** with the two values **civilian** and **military**. Based on these two values, there are two Vehicle specializations: **civilian Vehicle** and **military Vehicle**. Due to multiple inheritance, the result is an inheritance lattice where the number of the most detailed specializations would be 3 X 2 = 6 as follows: **civilian Car**, **civilian Aircraft**, **civilian Ship**, **military Car**, **military Aircraft**, and **military Ship**.

10.3.5 Classification-instantiation link

10.3.5.1 Classification-instantiation relation link

The fundamental structural relation classification-instantiation shall mean that a refineable, the class, classifies one or more refinees, the instances of the classification. The classification, which is an object class or a process class, is a source pattern for a thing connecting with one or more destination things, which are instances of the source thing's pattern, i.e. the qualities the pattern specifies acquire explicit values to instantiate the instance thing. This relation provides the modeller with an explicit mechanism for expressing the relationship between a class and its instances, which the provisioning of values creates.

NOTE 1 The use of the term instance when considering members of the instance set of a conceptual class are referred to as 'refinee instances' to distinguish them from 'operational instances' of an operating model. For every refinee instance, there are one or more operational instances possible.

NOTE 2 All OPM things expressed in a conceptual model are a class pattern for instances of that thing intended to occur during model evaluation or operation. By creating a thing in the conceptual model, the modeller is implying that at least one operational instance of that thing or a specialization of that thing may exist at some time during the system's operation.

If the class pattern includes an exhibition-characterization link specifying a refinee attribute with a permissible range of values, then the corresponding attribute value of each operational instance of a refinee instance of that class shall be within the value range specification of its class attribute feature.

Graphically, a small black circle inside an otherwise empty larger triangle with apex connecting by a line to the class thing and the instance things connecting by lines to the opposite base shall denote the classification-instantiation relation link.

The syntax of the classification-instantiation relation link between an object class and a single instance shall be: **Instance-object** is an instance of **Class-object.**

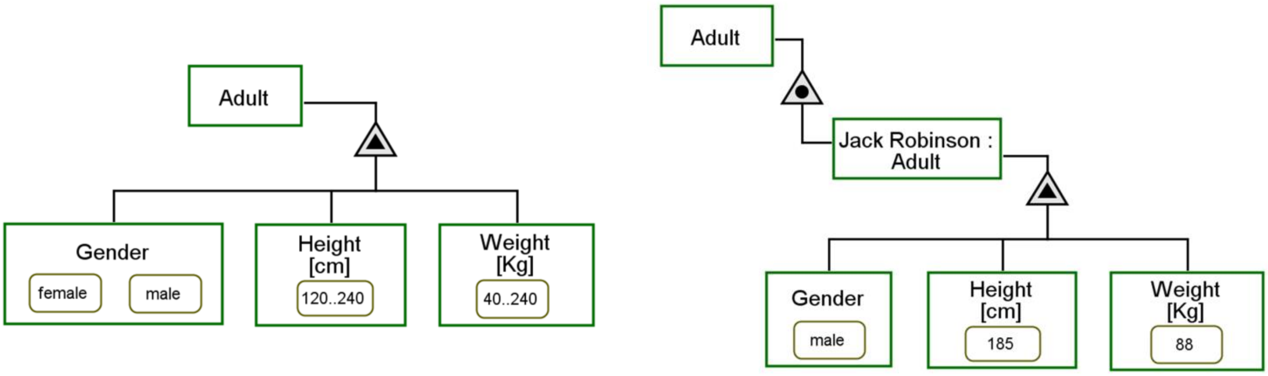
The syntax of the classification-instantiation relation link between a process class and a single instance shall be: **Instance-process** is an instance of **Class-process.**

The syntax of the classification-instantiation relation link between a process class and n instances shall be; **Instance-object**$_1$**, Instance-object**$_2$**, …, Instance-object**$_n$ are instances of **Class-object.**

The syntax of the classification-instantiation relation link between a process class and n instances shall be; **Instance-process**$_1$**, Instance-process**$_2$**, …, Instance-process**$_n$ are instances of **Class-process.**

NOTE 3 Since the number of instances of any class may not be known a priori and may vary during operation of the system, there is no distinction between complete and incomplete collections of destination things for the classification-Instantiation relation.

EXAMPLE 1 In Figure 26, **Adult** is a class with three attributes: **Gender**, with possible values **female** and **male**, **Height** in **cm**, with possible values **120..240**, and **Weight** in **kg**, with possible values **40..240**. **Jack Robinson** is an instance of **Adult**, with **Gender** value **male**, **Height** in **cm** value **185** and **Weight** in **kg** value **88**.

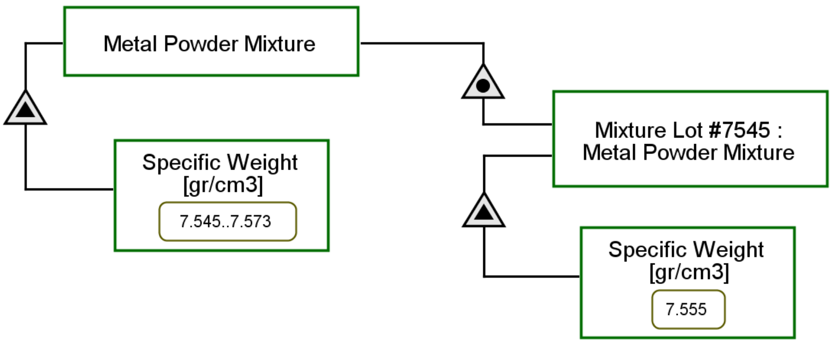


Adult exhibits **Gender, Height i**n **cm,** and **Weight** in **Kg.**
Gender of **Adult** can be **female** or **male.**
Height in **cm** of **Adult** ranges from **120 to 240.**
Weight in **Kg** of **Adult** range from **40 to 240.**

Jack Robinson is an instance of **Adult.**
Gender of **Jack Robinson** is **male.**
Height in **cm** of **Jack Robinson** is **185.**
Weight in **kg** of **Jack Robinson** is **88.**

Figure 26 — Classification-Instantiation with value range (class on left and instance on right)

EXAMPLE 2 The OPD on the left hand side of Figure 27 is a conceptual model of **Metal Powder Mixture**, indicating that its **Specific Weight** attribute value can range from 7.545 to 7.537 gr/cm^3. Figure 27 is an operational instance (runtime) model of **Metal Powder Mixture Instance**, indicating that its **Specific Weight** attribute value is 7.555 gr/cm^3. This value is within the allowable range.



Metal Powder Mixture exhibits **Specific Weight in gr/cm3.**
Specific Weight in **gr/cm3** of **Metal Powder Mixture** ranges from **7.545 to 7.537.**
Mixture Lot #7545 is an instance of **Metal Powder Mixture.**
Specific Weight in **gr/cm3** of **Metal Powder Mixture** is **7.555.**

Figure 27 — Attribute state as value: conceptual versus operational models

NOTE 4 The OPL sentence "**Mixture Lot #7545** exhibits **Specific Weight** in **gr/cm3.",** is not present in the OPL of Figure 27 because that sentence is implicit from the expressed fact "**Mixture Lot #7545** is an instance of **Metal Powder Mixture."**, and therefore **Mixture Lot #7545** inherits this attribute from **Metal Powder Mixture**.

10.3.5.2 Instances of object class and process class

An object class and a process class shall be two distinct kinds of classes. An instance of a class shall be an incarnation of a particular identifiable instance of that class with the same classification identifier.

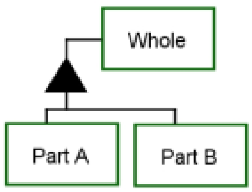
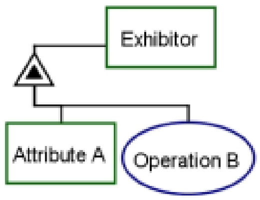
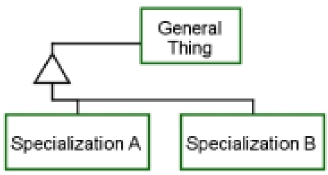
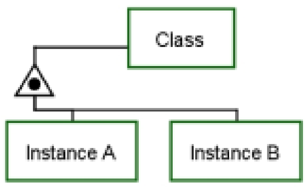
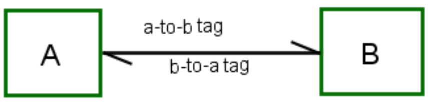
A single refinee object shall be an object instance, while the pattern of object, to which all of the instances adhere, shall be an object class, the refineable.

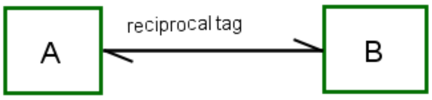
A process class shall be a pattern of happening (the sequence of subprocesses), which involves object classes that are members of the preprocess and postprocess object sets. A process occurrence, which follows this pattern and involves particular object instances in its preprocess and postprocess object sets, shall be a process instance. Hence, a process instance shall be a particular occurrence of a process class to which that instance belongs. Any process instance shall have associated with it a distinct set of preprocess and postprocess object instance sets.

NOTE The power of the process class concept is that it enables the modelling of a process as a template or a protocol for some transformation that a class of objects undergoes. That transformation includes neither the spatio-temporal framework nor the particular set of object instances with which the process instance associates.

10.3.6 Fundamental structural relation link and tagged structural link summary

Table 14 — Fundamental structural relations and link summary

Structural Relation Forward-Reverse (refineable-to-refinee; bold is the short name)	**OPD Symbol**	**OPL Sentence**	
		Forward refineable-to-refinee	**Reverse** (refinee-to-refineable)
Aggregation- Participation		**Whole** consists of **Part A** and **Part B**.	–
Exhibition-**Characterization**		**Exhibitor** exhibits **Attribute A** as well as **Operation B**.	–
Generalization- Specialization		–	**Specialization A** and **Specialization B** are **General Thing.**
Classification- Instantiation		–	**Instance A** and **Instance B** are instances of **Class.**
Unidirectional tagged [Unidirectional null tagged]		**Source tag-name Destination.** [**Source** relates to **Destination.**]	
Bidirectional tagged		**A a-to-b tag B.** **B b-to-a tag A.**	

Structural Relation Forward-Reverse (refineable-to-refinee; bold is the short name)	**OPD Symbol**	**OPL Sentence**	
		Forward refineable-to-refinee	**Reverse** (refinee-to-refineable)
Reciprocal tagged [Reciprocal null tagged]		**A** and **B** are **reciprocal tag.** [**A** and **B** are related**.**]	

10.4 State-specified structural relations and links

10.4.1 State-specified characterization relation link

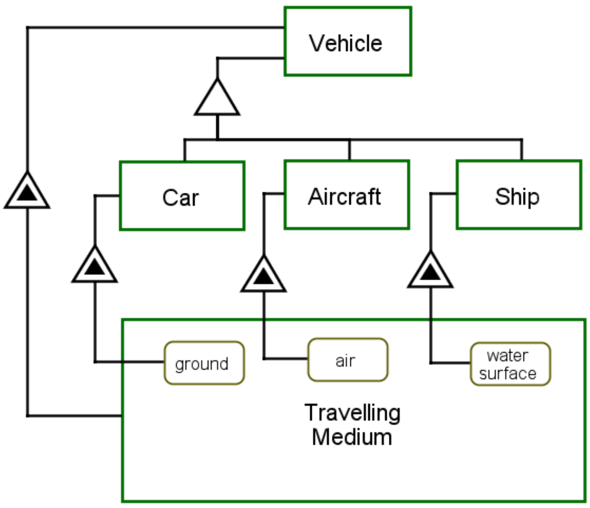
A state-specified characterization relation link shall be an exhibition-characterization relation link from a specialized object that exhibits an attribute value for a discriminating attribute of its generalization, meaning that the specialized object shall have only that value for the attribute it inherits.

Graphically, the exhibition-characterization relation link triangular symbol, with its apex connecting to the specialized object and its opposite base connecting to the value shown as a state, shall denote the state-specified characterization relation link.

NOTE While not necessary, the OPD will be more understandable if the exhibition-characterization link of the general with the discriminating attribute appears in the same OPD as well (see Figure 28).

The syntax of the state-specified characterization relation link shall be: **Specialized-object** exhibits **value-name Attribute-Name.**

EXAMPLE Using the state-specified characterization relation link, the OPD in in Figure 28 is significantly more compact than its equivalent OPD in Figure 25. Here, the discriminating attribute **Travelling Medium** of **Vehicle** with values **ground**, **air**, and **water surface** appears only once, as opposed to four times in Figure 25. The model for **Car**, **Aircraft**, and **Ship** are specializations of **Vehicle**, connecting each specialization with a state-specified characterization relation link to the corresponding **Travelling Medium** value of **ground**, **air**, and **water surface** respectively.



Vehicle exhibits **Travelling Medium.**
Travelling Medium of **Vehicle** can be **ground**, **air**, and **water surface.**
Car, **Aircraft**, and **Ship** are **Vehicles.**

Car exhibits **ground Travelling Medium.**
Aircraft exhibits **air Travelling Medium.**
Ship exhibits **water surface Travelling Medium.**

Figure 28 — State-specified characterization link example

10.4.2 State-specified tagged structural relations

10.4.2.1 State-specified tagged structural links

A state-specified tagged structural link shall be a tagged structural link between an object state or attribute value and another object, object state or attribute value, signifying a relation between these two things with the tag expressing the semantics of the relation. In case of a null tag, i.e. no explicit tag specification, the corresponding OPL shall use the default null tag (see 10.2.2.).

Three kinds of state-specified tagged structural links shall exist: source state-specified tagged structural link; destination state-specified tagged structural link; and, source-and-destination state-specified tagged structural link. Each kind shall include the unidirectional, bidirectional, and reciprocal tagged structural link, giving rise to seven kinds of state-specified tagged structural relation link and corresponding OPL sentences, which Table 15 summarizes.

10.4.2.2 Unidirectional source state-specified tagged structural link

A unidirectional source state-specified tagged structural link shall be a unidirectional tagged structural link from a specific state of the source object to a destination object without a state specification.

Graphically, an arrow with an open arrowhead connecting from a state of the source object to the destination object and a tag-name annotation near the shaft shall denote a unidirectional source state-specified tagged structural link.

The syntax of the unidirectional source state-specified tagged structural link OPL sentence shall be: **Specified-state source-object tag-name Destination-object.**

NOTE A null tag uses the default tag-name "relates to", not in bold, unless modified by the modeller.

10.4.2.3 Unidirectional destination state-specified tagged structural link

A unidirectional destination state-specified tagged structural link shall be a unidirectional tagged structural link from a source object without a state specification to a specific state of the destination object.

Graphically, an arrow with an open arrowhead connecting from a source object to a specific state of the destination object and a tag-name annotation near the shaft shall denote a unidirectional destination state-specified tagged structural link.

The syntax of the unidirectional destination state-specified tagged structural link OPL sentence shall be: **Source-object tag-name specified-state Destination-object.**

NOTE A null tag uses the default tag-name "relates to", not in bold, unless modified by the modeller.

10.4.2.4 Unidirectional source-and-destination state-specified tagged structural link

A unidirectional source-and-destination state-specified tagged structural link shall be a unidirectional tagged structural link from a specific state of a source object to a specific state of the destination object.

Graphically, an arrow with an open arrowhead connecting from a specific state of a source object to a specific state of the destination object and a tag-name annotation near the shaft shall denote a unidirectional source-and-destination state-specified tagged structural link.

The syntax of the unidirectional source-and-destination state-specified tagged structural link OPL sentence shall be: **Source-specified-state source-object tag-name destination-specified-state Destination-object.**

NOTE A null tag uses the default tag-name "relates to", not in bold, unless modified by the modeller.

10.4.2.5 Bidirectional source-or-destination state-specified tagged structural link

A bidirectional source-or-destination state-specified tagged structural link shall be a bidirectional tagged structural link with a specific state for either the source or destination object but not both.

Graphically, a line with harpoon shaped arrowheads on opposite sides at both ends of the link, one connecting to an object or object state and the other connecting to an object state or object respectively, shall denote a bidirectional tagged structural link. Each tag-name shall align on the side of the arrow with the harpoon edge sticking out of the arrowhead, unambiguously determining the direction in which each relation applies.

The syntax of the resulting bidirectional source-or-destination state-specified tagged structural link shall be two separate unidirectional tagged structural link OPL sentences, one for each direction with the corresponding state specifications.

10.4.2.6 Bidirectional source-and-destination state-specified tagged structural link

A bidirectional source-and-destination state-specified tagged structural link shall be a bidirectional tagged structural link with a specific state for both the source and destination object.

Graphically, a line with harpoon shaped arrowheads on opposite sides at both ends of the link, connecting a specific state of one object to a specific state of another object, shall denote a bidirectional tagged structural link. Each tag-name shall align on the side of the arrow with the harpoon edge sticking out of the arrowhead, unambiguously determining the direction to which each relation applies.

The syntax of the resulting bidirectional source-and-destination state-specified tagged structural link shall be two separate unidirectional source-and-destination tagged structural link OPL sentences, one for each direction with the corresponding state specifications and tag-names.

10.4.2.7 Reciprocal source-or-destination state-specified tagged structural link

A reciprocal source-or-destination tagged structural link shall be a bidirectional source-or-destination tagged structural link with a specific state for one of the involved objects but not both, and only one reciprocity-tag or no tag. In either case, reciprocity shall indicate that the tag of a reciprocal source-or-destination state-specified tagged structural link has the same semantics for each direction of the relation. When no tag appears, the default tag shall be "are related".

Graphically, a line with harpoon shaped arrowheads on opposite sides at both ends of the link, connecting a specific state of one object to another object without state specification and depicting only one tag-name aligning with the arrow, shall denote a reciprocal source-or-destination state-specified tagged structural link.

The syntax of the reciprocal source-or-destination state-specified tagged structural link with only one tag shall be either: **Source-specified-state Source-object** and **Destination-object** are **reciprocity-tag;** or**, Source-object** and **destination-specified-state Destination-object** are **reciprocity-tag.**

10.4.2.8 Reciprocal source-and-destination state-specified tagged structural link

A reciprocal source-and-destination tagged structural link shall be a bidirectional source-and-destination tagged structural link with a specific state for both involved objects, and only one reciprocity-tag or no tag. In either case, reciprocity shall indicate that the tag of a reciprocal source-and-destination state-specified tagged structural link has the same semantics for each direction of the relation. When no tag appears, the default tag shall be "are related".

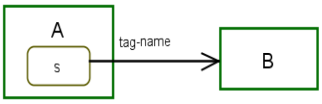
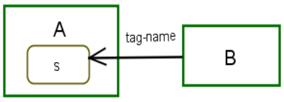
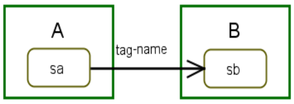
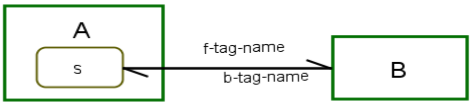
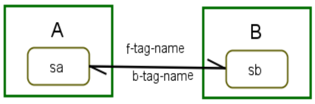
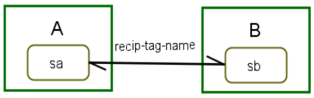
Graphically, a line with harpoon shaped arrowheads on opposite sides at both ends of the link, connecting a specific state of one object to a specific state of another object and depicting only one tag-name aligning with the arrow, shall denote a reciprocal source-and-destination state-specified tagged structural link.

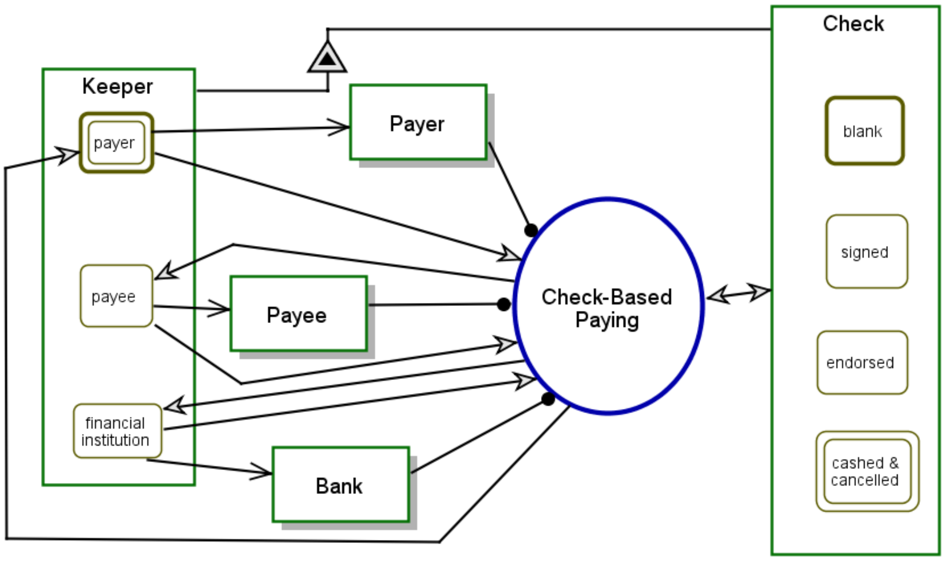
The syntax of the reciprocal source-and-destination state-specified tagged structural link with only one tag-name shall be: **Source-specified-state Source-object** and **destination-specified-state Destination-object** are **reciprocity-tag.**

The syntax of the reciprocal source-and-destination state-specified tagged structural link with no tag-name shall be: **Source-specified-state Source-object** and **destination-specified-state Destination-object** are related.

10.4.2.9 State-specified tagged structural link summary

Table 15 — State-specified structural relations and links summary

<table>
<tr><th>Source/ Destination
Directionality</th><th>source state-specified</th><th>destination state-specified</th><th>source-and-destination state-specified</th></tr>
<tr><td>unidirectional</td><td>S A tag-name B.</td><td>B tag-name s A.</td><td>Sa A tag-name sb B.</td></tr>
<tr><td>bidirectional</td><td colspan="2">S A f-tag-name B.
B b-tag-name s A.</td><td>Sa A f-tag-name sb B.
Sb B b-tag-name sa A.</td></tr>
<tr><td>reciprocal</td><td colspan="2">B and s A are recip-tag-name.</td><td>Sa A and sb B are recip-tag-name.</td></tr>
</table>

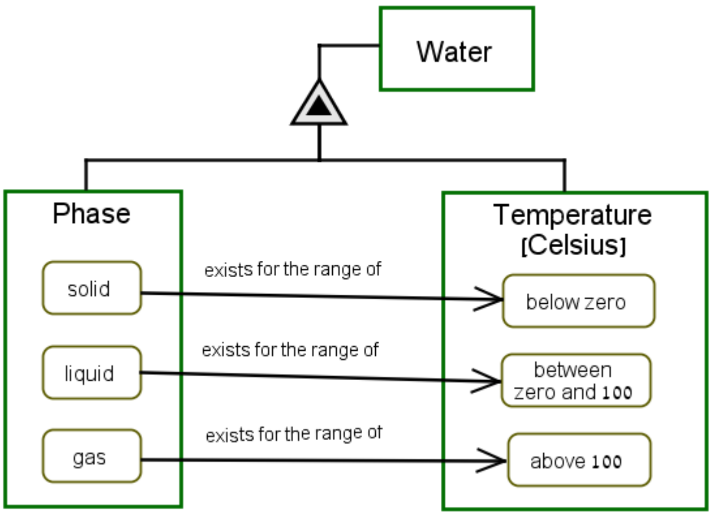


Check can be **blank**, **signed, endorsed**, or **cashed & cancelled.**
Check exhibits **Keeper.**
Keeper can be **payer**, **payee**, or **financial institution.**
Payer Keeper relates to **Payer.**
Payee Keeper relates to **Payee.**
Financial institution Keeper relates to **Bank.** (remaining OPL omitted)

Figure 29 — Associating attribute values with objects via state-specified structural link

EXAMPLE 1 In the OPD in Figure 29, **Keeper** is an attribute of **Check** with values **payer**, **payee**, and **bank**. Each of these values is also an object in its own right in the model. Three unidirectional, source-state-specified null-tagged structural links connect each value to its corresponding object. Note that there is no requirement that the name of the state or value be the same as the name of the related object, as demonstrated by **financial institution** and **Bank**.

EXAMPLE 2 In the OPD in Figure 30, each one of the three **Phase** values of **Water** is associated with its corresponding **Temperature** value range via three source-and-destination state-specified tagged structural links whose tag is "**exists for the range of**".



Water exhibits **Phase** and **Temperature** in **Celsius.**
Phase can be **solid**, **liquid**, or **gas.**
Temperature in **Celsius** can be **below zero**, **between zero and 100**, or **above 100.**
Solid Phase exists for the range of below zero Temperature in **Celsius.**
Liquid Phase exists for the range of between zero and 100 Temperature in **Celsius.**
Gas Phase exists for the range of above 100 Temperature in **Celsius.**

Figure 30 — Source-and-destination state-specified tagged structural link

11 Relationship cardinalities

11.1 Object multiplicity in structural and procedural links

Object multiplicity shall refer to a requirement or constraint specification, sometimes called a participation constraint, on the quantity or count of object operational instances associated with a link. Unless a multiplicity specification is present, each end of a link shall specify only one object operational instance. Multiplicity specifications may appear in the following situations:

(1) to specify multiple source or destination object operational instances for a tagged structural link of any kind;

(2) to specify a participant object with multiple operational instances in an aggregation-participation link, where a different participation specification may be attached to each one of the parts of the whole; and

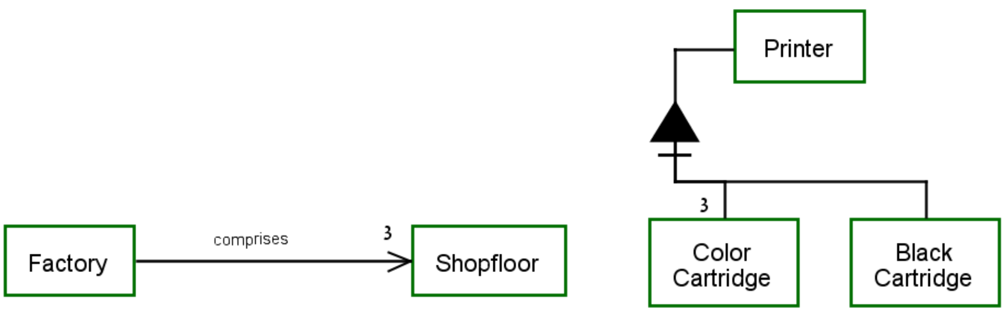
(3) to specify an object with multiple operational instances in a procedural relation.

The specification of object multiplicity may occur as integers or as parameter symbols that resolve to integer values during model execution and may include arithmetic expressions. The specification may include a range of values or a set of value ranges.

Graphically, an integer, a range of integers, a parameter symbol, a range of parameter symbols, or set of integers or parameter symbols, any of which may appear as annotations near the link end to which it applies, shall denote object multiplicity.

The syntax of an OPL sentence that includes an object with multiplicity shall include the object multiplicity preceding the object name, with the object name appearing in its plural form if the cardinality specifies more than one operational instance is possible. The following EXAMPLES present some of the many uses of object multiplicity on OPL sentences.

EXAMPLE Figure 31 shows in the left OPD a participation constraint on the destination end of a unidirectional tagged structural link. On the right OPD is a participation constraint on the destination (part) end for one of two objects of an aggregation-participation link.



Factory comprises **3 Shopfloors.**

Printer consists of **3 Color Cartridges, Black Cartridge**, and other parts**.**

Figure 31 — Object multiplicity examples

Object multiplicity may be a parameter or a range of parameters or a set of two or more ranges of numbers and/or parameters separated by a comma. A range shall be indicated as q_{min} **..** q_{max} and shall be closed, i.e. include the boundaries q_{min} and q_{max}. In OPL, the expression of the range symbol "**..**" shall be "through" and the expression of the comma that separates two adjacent ranges shall be "or".

The specification of object multiplicity may occur as an optionality parameter using the range symbol, the asterisk symbol and the question mark symbol in the following manner:

– "0..1" shall mean zero or one, using the question mark (?) annotation near the object to which it applies with an OPL syntax of "an optional " immediately preceding the object;

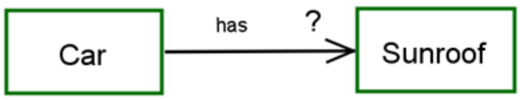
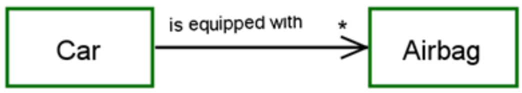
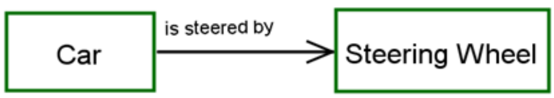
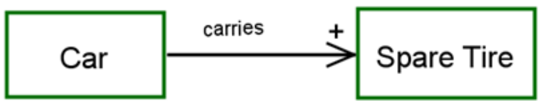
– "0..*" shall mean zero or more, using the asterisk symbol (*)annotation near the object to which it applies with the OPL syntax of "optional " immediately preceding the object, and

– "1..*" shall mean one or more, using the plus symbol (+) annotation near the object to which it applies with OPL syntax of "at least one " immediately preceding the object

NOTE 1 The range symbol ".." has two uses in multiplicity specification, one as a separator between two boundary values, e.g. qmin .. qmax, with interpretation of "through" and one as separator between optional values, e.g. "0..*" , with interpretation of "or".

NOTE 2 Care is necessary when specifying cardinality constraints so that the constraint applies to the object as specified and not a property of that object. If the object has a unit of measure, then multiplicity refers to the count of single units of that measure, e.g. 32 **Water** in millilitres.

Table 16 - Link optionality summary

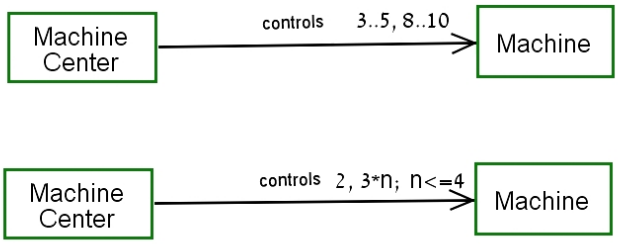
Lower & Upper Bounds q_{min} .. q_{max}	**Participation Constraint Symbol & OPL Phrase**	**OPD Example & Corresponding OPL Sentence**
0..1	? an optional	 **Ca**r has an optional **Sunroof.**
0..*	* optional (+ plural)	 **Car** is equipped with optional **Airbags.**
1..1	(none)	 **Car** is steered by **Steering Wheel.**
1..*	+ at least one	 **Car** carries at least one **Spare Tire.**

11.2 Object multiplicity expressions and constraints

Object multiplicity may include arithmetic expressions, which shall use the operator symbols "+", "–", "*", "/", "(", and ")" with their usual semantics and shall use the usual textual correspondence in the corresponding OPL sentences.

An integer or an arithmetic expression may constrain object multiplicity. Graphically, expression constraints shall appear after a semicolon separating them from the expression that they constrain and shall use the equality/inequality symbols "=", "<", ">", "<=", and ">=", the curly braces "{" and "}" for enclosing set members, and the membership operator "in" (element of, $\in$), all with their usual semantics. The corresponding OPL sentence shall place the constraint phrase in bold letters after the object to which the constraint applies in the form ", where constraint".

EXAMPLE 1

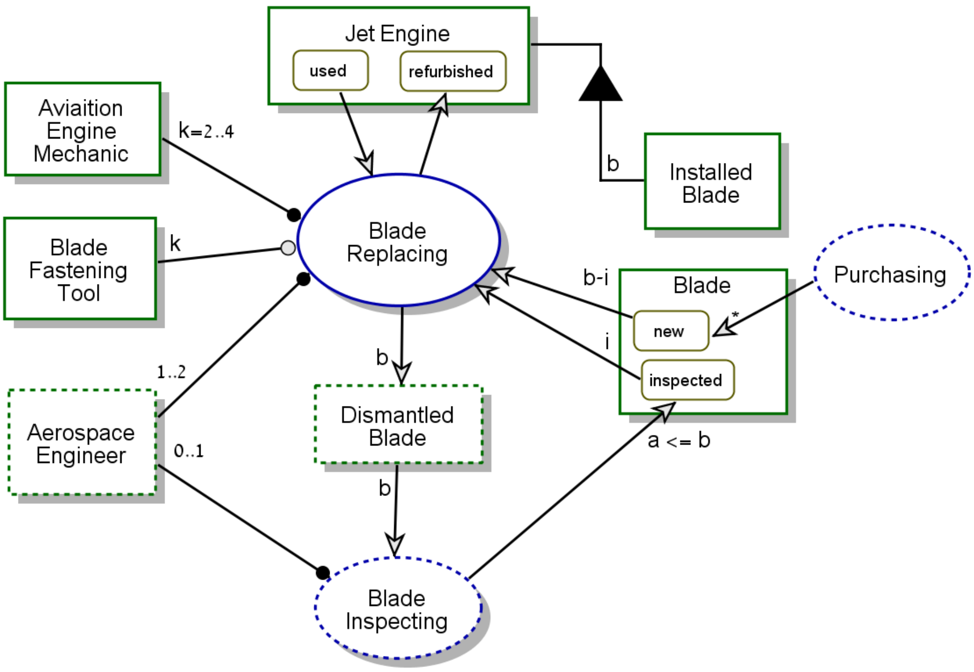


Machine Center controls 3 to **5** or **8** to **10 Machines.**
Machine Center controls 2 or **3*n Machines**, where **n<=4.**

Figure 32 — Object multiplicity examples with ranges and parameters

EXAMPLE 2 Figure 33 models a **Blade Replacing** system in which a **Jet Engine** has **b Installed Blades**. Two to four (a number set to **k**) **Aviation Engine Mechanics** handle the **Blade Replacing** process, for which they use **k Blade Fastening Tools**. Also, one or two **Aerospace Engineers** handle the **Blade Replacing** process. This process yields **b Dismantled Blades**, which undergo **Blade Inspecting**, an environmental process that yields **a** (which is at most **b**) of **Inspected Blades**. The process consumes a total of **b Blades**, with **i inspected** and **b–i new**. Any number of **new Blades** can be obtained by **Purchasing** them.

k=2 to 4 Aviation Engine Mechanics handle **Blade Replacing.**



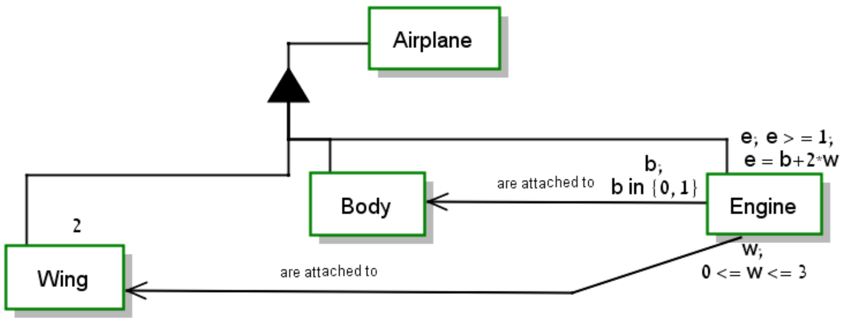
Jet Engine can be **used** or **refurbished.**
Jet Engine consists of **b Installed Blades.**
1 to 2 Aerospace Engineers handle **Blade Replacing.**
An optional **Aerospace Engineer** handles **Blade Inspecting.**
Blade can be **inspected** or **new.**
Blade Replacing requires **k Blade Fastening Tools.**
Blade Replacing changes **Jet Engine** from **used** to **refurbished.**

Blade Replacing consumes **i inspected Blades** and **b – i new Blades.**
Blade Replacing yields **b Dismantled Blades.**
Blade Inspecting consumes **b Dismantled Blades.**
Blade Inspecting yields **a <= b inspected Blades.**
Purchasing yields many **new Blades.**

Figure 33 — Object multiplicity: arithmetic expressions and constraints example

If an object multiplicity parameter has more than one constraint, they shall appear as a semicolon-separated list of constraints following the parameter. Any constraint may include any object multiplicity parameter appearing in the model. Parameter names shall be unique for the entire system model.

EXAMPLE 3 Figure 34 depicts a way to specify parameterized participation constraints in an OPD and the corresponding OPL sentences.



Airplane consists of **Body**, **2 Wings**, and **e Engines**, where **e >= 1, e = b+2*w.**
b Engines are attached to Body, where **b in {0, 1}.**
w Engines are attached to Wing, where **0 <= w <= 3.**

Figure 34 — Multiple parameterized constraints example

NOTE 1 Aggregation-participation is the only fundamental structural relation for which participation constraints apply.

NOTE 2 Expressing multiplicity of processes does not use participation constraints. Rather, expressing sequential repetition of the same process uses a recurrent process with a counter for the number of iterations. Parallel synchronous processes or asynchronous processes within an in-zoomed process provide other iteration mechanisms.

11.3 Attribute value and multiplicity constraints

The expression of object multiplicity for structural and procedural links specifies integer values or parameter symbols that resolve to integer values. In contrast, the values associated with attributes of objects or processes may be integer or real values, or parameter symbols that resolve to integer or real values, as well as character strings and enumerated values.

NOTE 1 Real values accommodate expression using the unit of measure associated with the object.

Graphically, a labelled, rounded-corner rectangle placed inside the attribute to which it belongs shall denote an attribute value with the value or value range (integers, real numbers, or string characters) corresponding to the label name. In OPL text, the attribute value shall appear in bold face without capitalization.

The syntax for an object with an attribute value OPL sentence shall be: **Attribute** of **Object** is **value**.

The syntax for an object with an attribute value range OPL sentence shall be: **Attribute** of **Object** range is **value-range**.

NOTE 2 Attribute value range has the same expressiveness applicable for object multiplicity, except optionality.

A structural or a procedural link connecting with an attribute that has a real number value may specify a relationship constraint, which is distinct from an object multiplicity.

Graphically, an attribute value constraint is an annotation by a number, integer or real, or a symbol parameter, near the attribute end of the link and aligning with the link.

12 Logical operators: AND, XOR, and OR

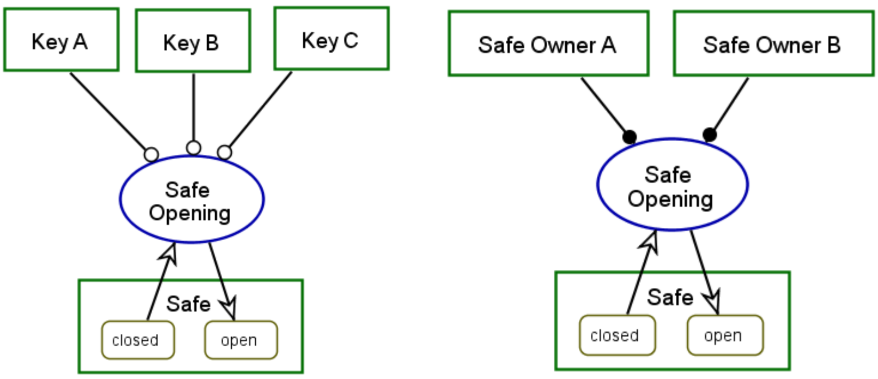
12.1 Logical AND procedural links

A group of two or more procedural links of the same kind that originate from, or arrive at, the same process shall have the semantics of logical AND.

Graphically, the links with AND semantics do not touch each other on the process contour.

The syntax of links with AND semantics shall be a phrase using "and" conjunction in a single OPL sentence rather than separate sentences for each link

EXAMPLE 1 Figure 35 (right), the **Safe Opening** process requires both **Safe Owner A** and **Safe Owner B.** In Figure 35 (left), opening the **Safe** requires all three keys.

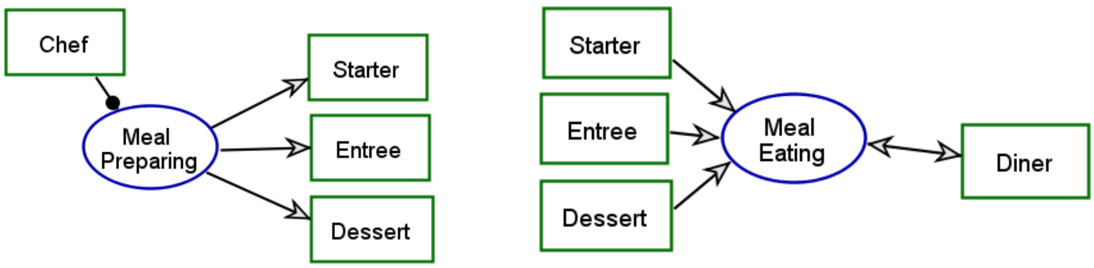


Safe can be **closed** or **open.**
Safe Opening requires **Key A, Key B,** and **Key C.**
Safe Opening changes **Safe** from **closed** to **open.**

Safe can be **closed** or **open.**
Safe Owner A and **Safe Owner B** handle **Safe Opening.**
Safe Opening changes **Safe** from **closed** to **open.**

Figure 35 — Logical AND for Agent and Instrument Links

EXAMPLE 2 In Figure 36 (left), **Meal Preparing** yields all three of the dishes. In Figure 36 (right), **Meal Eating** consumes all three dishes.

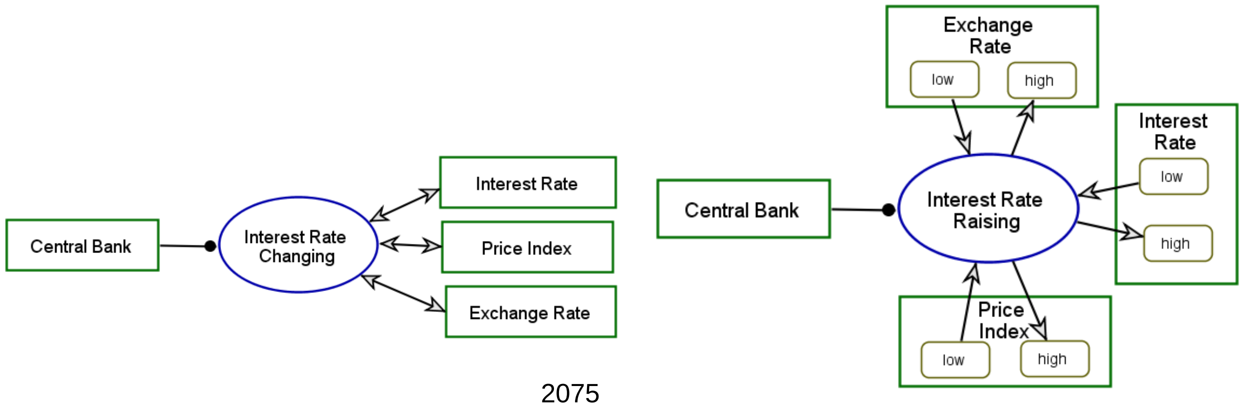


Chef handles **Meal Preparing.**
Meal Preparing yields **Starter, Entree,** and **Dessert.**

Meal Eating affects **Diner.**
Meal Eating consumes **Dessert, Entree,** and **Starter.**

Figure 36 — Logical AND for Result and Consumption Links

EXAMPLE 3 In the OPD on the left of Figure 37, **Interest Rate Changing** affects the three objects **Exchange Rate**, **Price Index**, and **Interest Rate**. In the OPD on the right, all three effects of **Interest Rate Raising** on **Exchange Rate**, **Price Index**, and **Interest Rate** are explicit via three pairs of input-output-specified effect links.



Central Bank handles **Interest Rate Changing**.
Interest Rate Changing affects **Exchange Rate, Price Index,** and **Interest Rate.**

Central Bank handles **Interest Rate Changing.**
Interest Rate can be **high** or **low.**
Price Index can be **low** or **high.**
Exchange Rate can be **high** or **low.**
Interest Rate Raising changes **Exchange Rate** from **low** to **high, Price Index** from **low** to **high,** and **Interest Rate** from **low** to **high.**

Figure 37 — Logical AND for Effect Link and Input-Output Links Pair

NOTE See 13 for impacts of path labels on AND syntax.

12.2 Logical XOR and OR procedural links

A group of two or more procedural links of the same kind that originate from a common point, or arrive at a common point, on the same object or process shall be a link fan. A link fan shall follow the semantics of either a XOR or an OR operator. The link fan end that is common to the links shall be the convergent link end. The link end that is not common to the links shall be the divergent link end.

The XOR operator shall mean that exactly one of the things at the divergent link end of the link fan exists. If the divergent link end has objects, then only one exists. If the divergent link end has processes, then only one occurs.

NOTE This use of the XOR operator in OPM is different to some binary XOR operator interpretations, where the output is 1 for an odd number of inputs and 0 for an even number of inputs.

Graphically, a dashed arc across the links of the link fan with the arc focal point at the convergent end-point of contact shall denote the XOR operator.

The syntax of a link fan of n things with XOR semantics shall be a single OPL sentence containing a phrase of the form: exactly one of $Thing_1$, $Thing_2$,…, and $Thing_n$...

The OR operator shall mean that at least one of the two or more things at the divergent end of the link fan exists. If the divergent link end has objects, then at least one object exists. If the divergent end has processes, then at least one process occurs.

Graphically, two concentric dashed arcs across the links of the link fan with the focal point at the convergent end-point of contact shall denote the OR operator.

The syntax of a link fan of n things with OR semantics shall be a single OPL sentence containing a phrase of the form: at least one of $Thing_1$, $Thing_2$,…, and $Thing_n$...

EXAMPLE In the OPD on the right of Figure 38, using XOR, exactly one of **Safe Owner A** and **Safe Owner B** must be present in order for **Safe Opening** to occur. In the OPD on the left, using OR, at least one of **Safe Owner A** and **Safe Owner B** must be present in order for **Safe Opening** to occur. The link fan here is convergent and consists of two agent links.

Exactly one of **Safe Owner A** and **Safe Owner B** handles **Safe Opening.**

At least one of **Safe Owner A** and **Safe Owner B** handles **Safe Opening.**

Figure 38 — Logical OR (left) and logical XOR (right) examples of Agent link

12.3 Diverging and converging XOR and OR links

Table 17 shows that when the source things are objects and the destination thing is a process, the consumption link fan is converging, while when the source things are processes and the destination thing is an object, the result link fan is converging.

Table 17 — Summary of XOR and OR converging consumption and result links

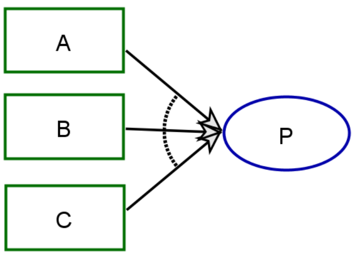
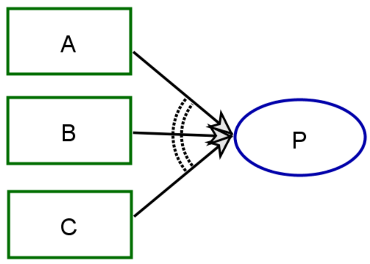
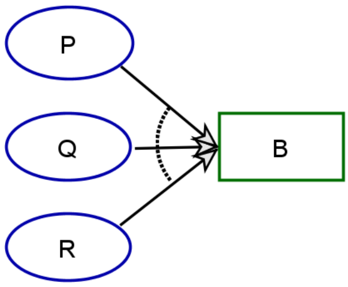
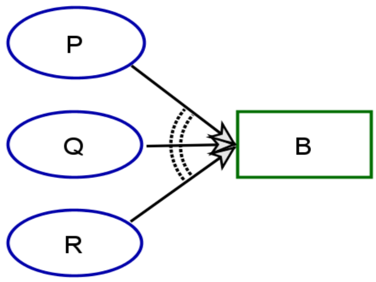
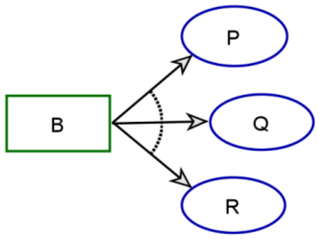
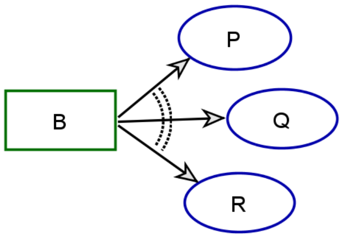
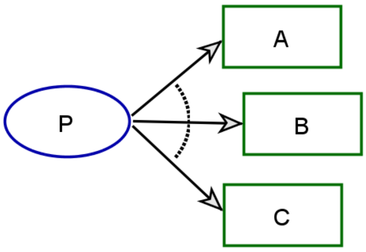
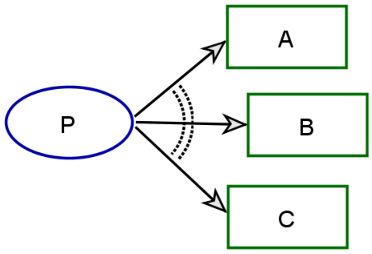
	XOR	**OR**
Converging consumption link fan	 **P** consumes exactly one of **A, B ,** or **C.**	 **P** consumes at least one of **A, B ,** or **C.**
Converging result link fan	 Exactly one of **P, Q,** or **R** yields **B.**	 At least one of **P, Q,** or **R** yields **B.**

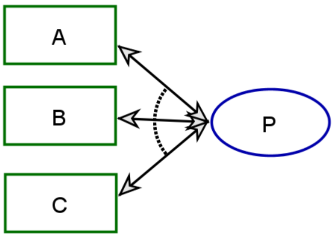
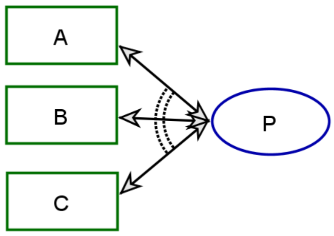
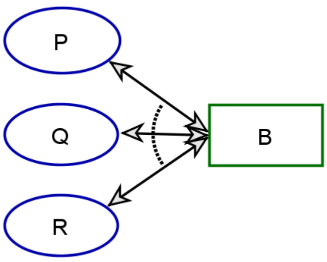
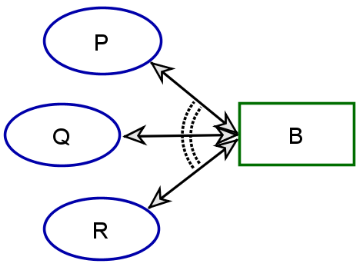
Table 18 shows that when the source thing is an object and the destination things are processes, the consumption link fan shall be diverging, while when the source thing is a process and the destination things are objects, the result link fan shall be diverging.

Table 18 — Summary of XOR and OR diverging consumption and result link fans

	XOR	OR
Diverging consumption link fan	 Exactly one of **P, Q** **,** or **R** consumes **B.**	 At least one of **P, Q** **,** or **R** consumes **B.**
Diverging result link fan	 **P** yields exactly one of **A, B,** or **C.**	 **P** yields at least one of **A, B,** or **C.**

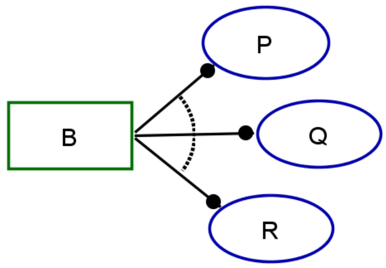
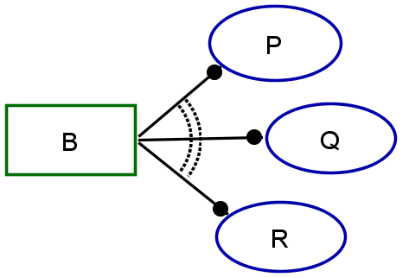
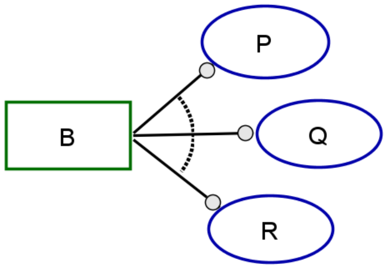
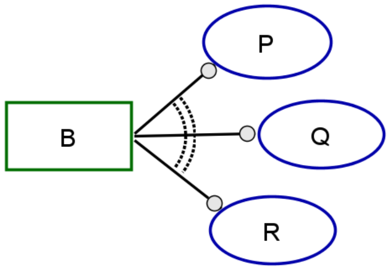
Since an effect link is bidirectional, the things linked by an effect link fan are both source and destination at the same time, voiding the definitions of convergent and divergent link fans. Instead, as Table 19 shows, the distinction shall occur with respect to multiple objects or multiple processes that a link fan connects.

Table 19 — Summary of XOR and OR joint effect link fans

	XOR	OR
Multiple objects effect link fan	 **P** affects exactly one of **A**, **B**, or **C.**	 **P** affects at least one of **A**, **B**, or **C.**
Multiple processes effect link fan	 Exactly one of **P**, **Q**, or **R** affects **P.**	 At least one of **P**, **Q**, or **R** affects **P.**

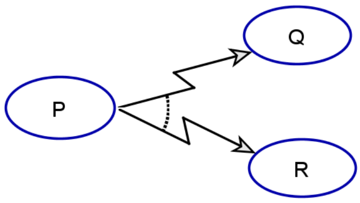
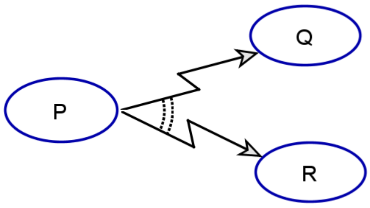
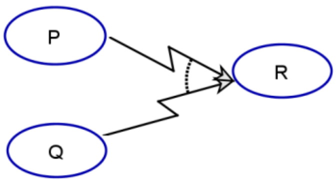
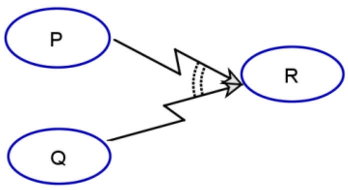
Since an enabler is an object, as shown in Table 20, both agent and instrument link fans shall be divergent with multiple processes as targets.

Table 20 — Agent and instrument link fans

	XOR	OR
Agent link fan	 **B** handles exactly one of **P**, **Q**, or **R.**	 **B** handles at least one of **P**, **Q,** or **R.**
Instrument link fan	 Exactly one of **P, Q,** or **R** requires **B.**	 At least one of **P, Q, or R** requires **B.**

Invocation link fans may be diverging or converging for both XOR and OR, as shown in Table 21.

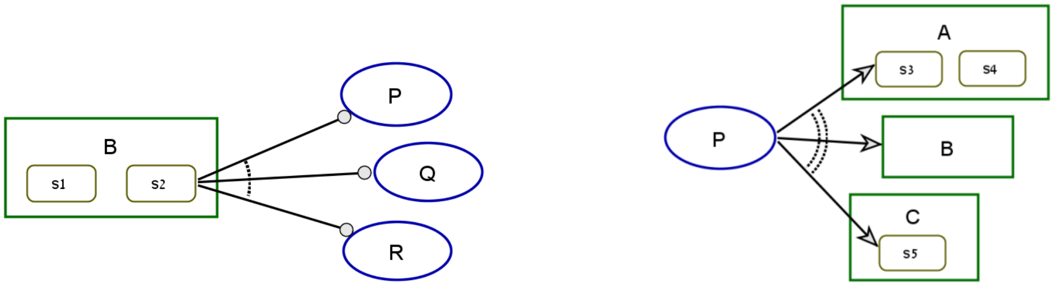
Table 21 — Invocation link fans

	XOR	OR
Diverging invocation link fan	 **P** invokes exactly one **Q** or **R.**	 **P** invokes at least one of **Q** or **R.**
Converging invocation link fan	 Exactly one of **P** or **Q** invokes **R.**	 At least one of **P** or **Q** invokes **R.**

12.4 State-specified XOR and OR link fans

Each one of the link fans in 12.3 shall have a corresponding state-specified version, where the source and destination may be specific object states or objects without a state specification. Combinations of state-specified and stateless links as destinations of a link fan may occur.

EXAMPLE Figure 39 shows on the left a XOR state-specified instrument link fan and on the right an OR mixed result link fan where the links are state-specified for objects **A** and **C** but not for **B**.



Exactly one of **P, Q,** or **R** requires **s2 B.**   **P** yields at least one of **s3 A, B,** or **s5 C.**

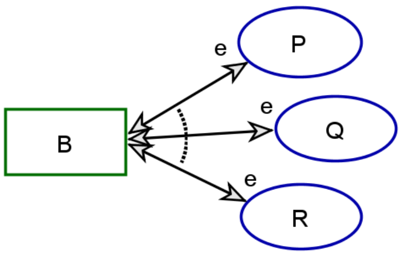
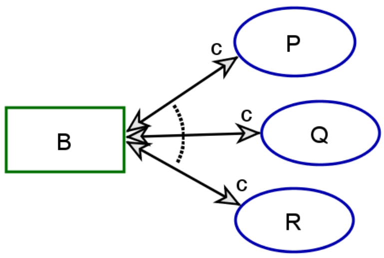
Figure 39 — State-specified XOR and OR link examples

12.5 Control-modified link fans

Each one of the XOR link fans for consumption, result, effect, and enabling links and their state-specified versions shall have a corresponding control-modified link fan: an event link fan and a condition link fan.

Table 22presents the event and condition effect link fans, as representatives of the basic (non-state-specified) links version of the modified link fans.

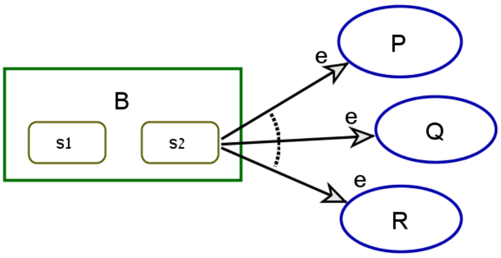
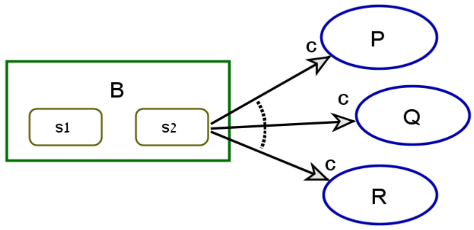
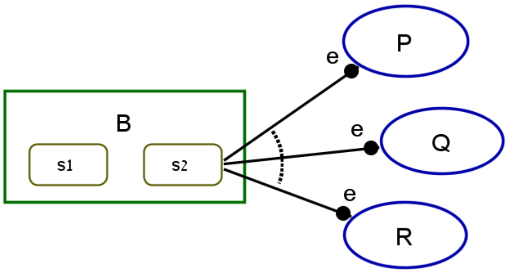
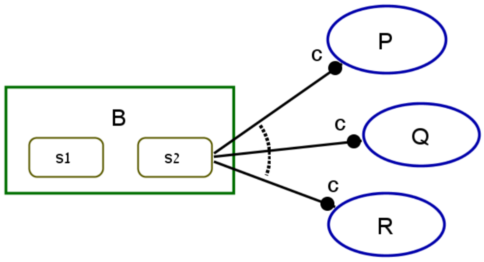
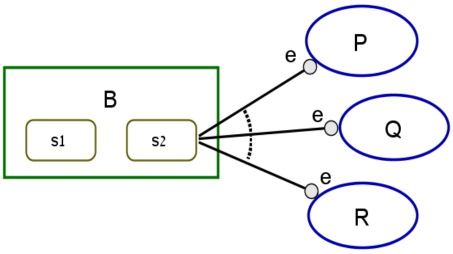
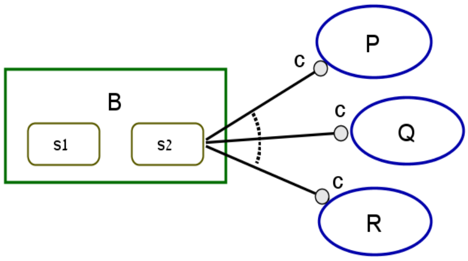
Table 22 — Event and condition effect link fans

Event	Condition
 **B** initiates exactly one of **P, Q,** or **R,** which affects the occurring process**.**	 Exactly one of **P, Q,** or **R** occurs if **B** exists**,** in which case the occurring process affects **B,** otherwise these processes are skipped**.**

12.6 State-specified control-modified link fans

Each one of the control-modified link fans, except the control-modified effect link fan, shall have a corresponding state-specified control-modified link fan. Since these state-specified versions are more complicated than their non-state-specified version, Table 23 presents the OPD and OPL of the state-specified versions and the corresponding stateless version below for each state-specified version.

Table 23 — State-specified and stateless control-modified link fans

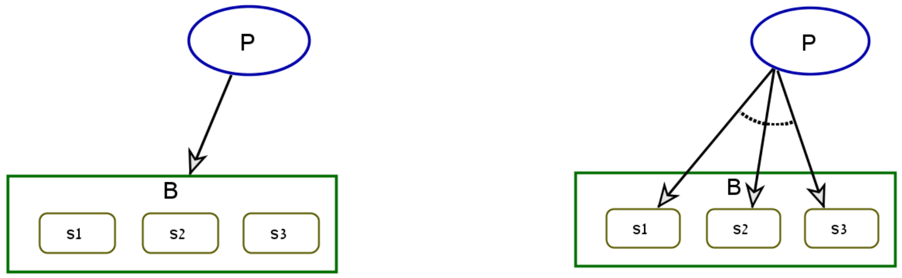
Link fan kind	Event Control modifier	Condition Control modifier
Consumption link fan	 **S2 B** initiates and handles exactly one of **P, Q, or R,** which consumes the initiated process. *The stateless case:* **B** initiates exactly one of **P, Q,** or **R,** which consumes the initiated process.	 Exactly one of **P, Q,** or **R** occurs if **B** is **s2,** in which case the occurring process consumes **B,** otherwise these processes are skipped. *The stateless case:* Exactly one of **P, Q, or R** occurs if **B** exists, in which case the occurring process consumes **B,** otherwise these processes are skipped.
Agent link fan	 **S2 B** initiates and handles exactly one of **P, Q,** or **R.** *The stateless case:* **B** initiates and handles exactly one of **P, Q,** or **R.**	 **B** handles exactly one of **P, Q,** or **R** if **B** is **s2,** otherwise these processes are skipped. *The stateless case:* **B** handles exactly one of **P, Q,** or **R** if **B** exists, otherwise these processes are skipped.
Instrument link fan	 **S2 B** initiates exactly one of **P, Q,** or **R,** which requires **s2 B.** *The stateless case:* **B** initiates exactly one of **P, Q,** or **R,** which requires **s2 B.**	 Exactly one of **P, Q,** or **R** requires that **B** is **s2,** otherwise these processes are skipped. *The stateless case:* Exactly one of **P, Q,** or **R** requires that **B** exists, otherwise these processes are skipped.

Each XOR link fan in Table 22 and in Table 23 shall have its OR counterpart (designated by a double-dotted arc) with a corresponding OPL sentence in which the reserved phrase "at least" replaces "exactly".

12.7 Link probabilities and probabilistic link fans

A process **P** with a result link that yields a stateful object **B** with n states, $\mathbf{s_1}$ through $\mathbf{s_n}$, without specifying a particular state shall mean that the probability of generating **B** at any one particular state shall be 1/n. In this case, the single result link to the object shall replace the result link fan to each of its states.

EXAMPLE 1 In the left OPD of Figure 40, the result link from **P** to **B**, which has three states, means that P will create B with equal probability, Pr = **1/3,** for creation at each state. The right OPD of Figure 40 shows the more cumbersome way to express the same situation.



B can be **s1, s2,** or **s3.**

P yields **B.**

B can be **s1, s2,** or **s3.**

P yields exactly one of **s1 B, s2 B,** or **s3 B.**

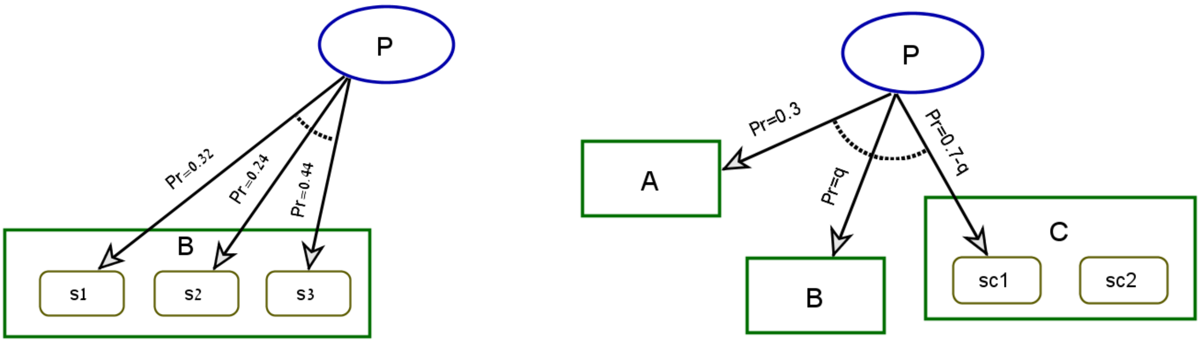
Figure 40 — Equivalence between result link and a set of XOR state-specified result links

Generally, probabilities of following a specific link in a link fan are not equal. Link probability may be a property value assigned to a link in a XOR diverging link fan that specifies the probability of following that particular link among the possible links in the fan link. A probabilistic link fan shall be a link fan with annotations on each fan link for its probability property, where the sum of the probabilities shall be exactly 1.

Graphically, along each fan link with a probability property an annotation shall appear in the form Pr=p, where p is the link probability numeric value or a parameter, which denotes the probability of the system execution control to select and follow that particular link of the fan.

The corresponding OPL sentence shall be the XOR diverging link fan sentence without link probabilities omitting the phrase "exactly one of…" and the phrase "…with probability p" following each participating thing name with a probability annotation "Pr=p".

EXAMPLE 2 Figure 41 shows two probabilistic state-specified object creation examples and their deterministic analogues. In the OPD on the left, process **P** can create object **B** in three possible states, **s1**, **s2**, or **s3**, with corresponding probabilities 0.32, 0.24, and 0.44 indicated along each result link of the result link fan. In the OPD on the right, **P** can create one of the objects **A**, **B**, or **C** at state **sc1** with the probabilities indicated along each result link of the result link fan.



P yields **s1 B** with probability **0.32, s2 B** with probability **0.24,** or **s3 B** with probability **0.44.**

The analogous deterministic case:

P yields exactly one of **s1 B, s2 B,** or **s3 B.**

P yields **A** with probability **0.3, B** with probability **q,** or **sc1 C** with probability **0.7-q.**

The analogous deterministic case:

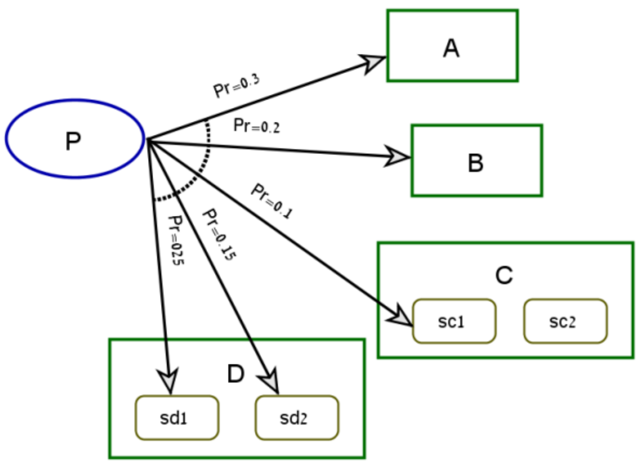
P yields exactly one of **A, B,** or **sc1 C.**

Figure 41 — Probabilistic state-specified object creation examples

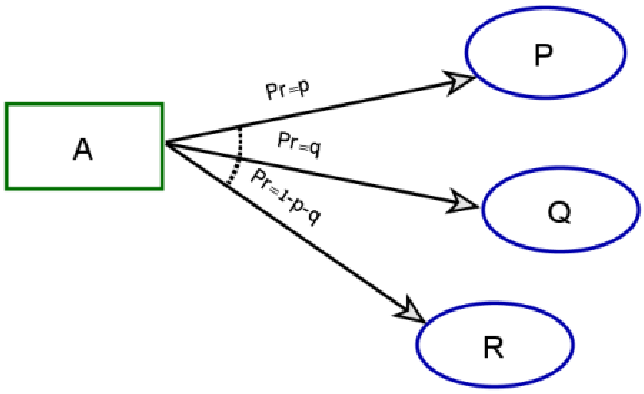
For a process **P** with a result link that yields a stateful object **B** with states $\mathbf{s_1}$ through $\mathbf{s_n}$, and with initial state $\mathbf{s_i}$, **P** shall create **B** at state $\mathbf{s_i}$ with probability 1.0. However, if **B** has m with m < n initial states, **P** shall create **B** at one of the initial states with probability 1/m.

For a probabilistic result link fan, any one of the resultees may be an object without or with a specified state. For all the link fans comprising other procedural link kinds (including those with the event and condition control modifiers), where the targets of the links in the link fan are processes, the source may be an object or a specified state of an object.

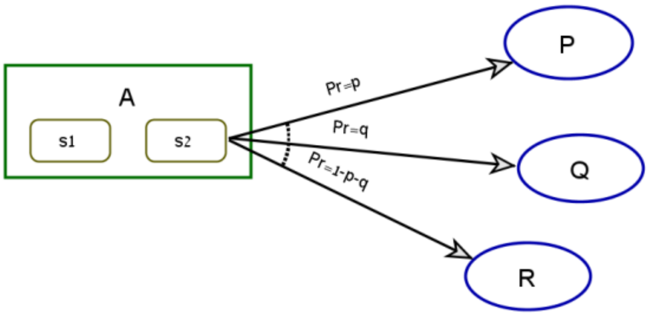
EXAMPLE 3 The OPD in the top of Figure 42 shows a probabilistic result link fan in which **P** yields, with specified probabilities, one of the objects **A** or **B**, or **C** at state **sc1**, or **D** at state **sd1** or **sd2**. The OPD in the middle of Figure 42 shows a probabilistic consumption link fan in which **A** is consumed, with specified probabilities, by one of the processes **P** or **Q** or **R**. The OPD in the bottom expresses the same, with the additional fact that **A** must be at state **s2**.



P yields **A** with probability **0.3, B** with probability **0.2, sc1 C** with probability **0.1, sd1 D** with probability **0.25,** or **sd2 D** with probability **0.15.** ....



P with probability **p, Q** with probability **q,** or **R** with probability **1-p-q** consumes **A.**



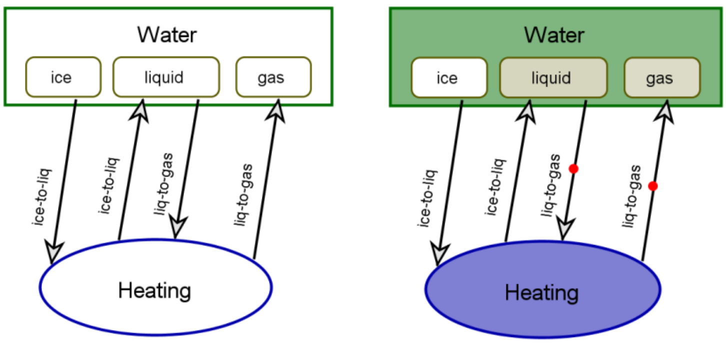
P with probability **p, Q** with probability **q,** or **R** with probability **1-p-q** consumes **s2 A.**

Figure 42 — Objects with and without specified states as resultees and consumees of a probabilistic link fan

13 Execution path and path labels

A path label shall be a link property and corresponding annotation aligning a pair of procedural links. When the process precondition involves an object with path label link connections, and the postprocess object set has more than one possibility for destination object, the appropriate postprocess object set destination shall be the one obtained using a link with the same path label as that used by the preprocess object set.

EXAMPLE 1 In Figure 43, there are two output links: one from **Heating** to the state **liquid** of **Water** and the other to state **gas**. When entering **Heating** from state **ice**, it is not clear whether the result state is **liquid** or **gas**. The path labels along the procedural links, resolve this dilemma by uniquely determining the appropriate link on process exit, as shown by the animated simulation on the left.



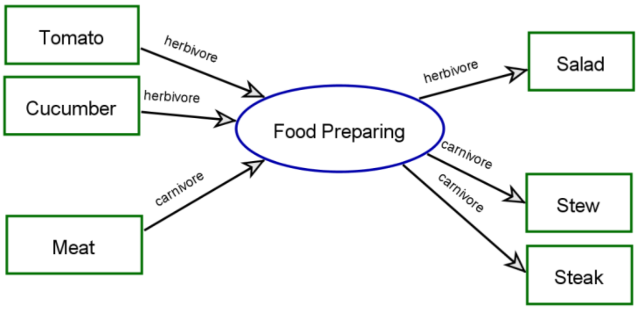
Water can be **ice**, **liquid**, or **gas.**

Following path ice-to-liq**, Heating** changes **Water** from **ice** to **liquid.**
Following path liq-to-gas, **Heating** changes **Water** from **liquid** to **gas.**

Figure 43 — Execution path and path labels

NOTE A path label is a label on a procedural link that removes the ambiguity arising from multiple outgoing procedural links by specifying that the link to follow is the one with the same label as the one initiating the process.

EXAMPLE 2 Figure 44 demonstrates the use of path labels on consumption and result links, followed by the OPL paragraph.



Following path **carnivore, Food Preparing** consumes **Meat.**
Following path **herbivore, Food Preparing** consumes **Cucumber** and **Tomato.**
Following path **carnivore, Food Preparing** yields **Stew** and **Steak.**
Following path **herbivore, Food Preparing** yields **Salad.**

Figure 44 — Path labels demonstrated on consumption and result links

14 Context management with Object-Process Methodology

14.1 Completing the system diagram

The definition of system purpose, scope, and function in terms of boundary, stakeholders, preconditions and postconditions shall be the basis for determining whether other elements, including environmental things, should appear in the model.

The System Diagram (SD) shall be an OPD that models:

— the stakeholders, in particular the beneficiaries;

— a process to convey the functional value the beneficiary expects to receive; and

— other environmental and systemic things necessary to create a succinct corresponding OPL paragraph.

The corresponding OPL paragraph should provide the situational context for the system's operation.

Expression of the functional value may be:

— explicit, by identifying the source input and destination output states of the beneficiary or the initial and final values of one or more of its attributes, or

— implicit, by indicating that the beneficiary is affected by the system's function.

The SD should contain only the central, important things – those things indispensable for understanding the function and context of the system. The modeller shall use OPM's refinement mechanisms to expose gradually the detail concerning the things that are the content of the SD.

EXAMPLE In a **Manufacturing Facility**, the **Beneficiary** has developed and deployed a **Preventive Maintenance System**. The function of the system, **Preventive Maintenance Executing**, changes the **Downtime** attribute of the **Manufacturing Facility** from "high" to "low". This change adds functional value to the **Manufacturing Facility**, as it has more up-time to manufacture products and increase sales and revenues at the cost of investing in developing and operating the **Preventive Maintenance System**.

14.2 Achieving model comprehension

14.2.1 OPM refinement-abstraction mechanisms

OPM shall provide abstracting and refining mechanisms to manage the expression of model clarity and completeness. These mechanisms make possible the specification of contextualized model segments as separate, yet interconnected OPDs, which, taken together, should provide a model of the functional value providing system. These mechanisms shall enable presenting and viewing the modelled system, and the elements it contains, in various contexts that are interrelated by the common objects, processes and relations. The set of clearly specified and compatible interconnected Object-Process Diagrams should completely specify the entire system to an appropriate extent of detail and provide a comprehensive representation of that system with a corresponding textual statement of the model in OPL.

The OPM refinement-abstraction mechanisms shall be the following three pairs: State expression and suppression, unfolding and folding, and in-zooming and out-zooming.

14.2.1.1 State expression and state suppression

Explicitly depicting the states of an object in an OPD may result in a diagram that is too crowded or busy, making it hard to read or comprehend.

OPM shall provide an option for state suppression, which suppresses the appearance of some or all the states of an object as represented in a particular OPD when those states are not necessary in that OPD's context.

The inverse of state suppression shall be state expression, which exposes information concerning possible object states. The OPL corresponding to an OPD shall express the states of the objects only as the OPD depicts.

In OPM the modeller may suppress any subset of states. However, the complete set of object states for an object shall be the union of the states of that same object appearing in all of the OPDs of the entire OPM model.

Graphically, the annotation indicating that an object presents a proper subset (i.e. at least one but not all) of its states, shall be a small state suppression symbol in the object's right bottom corner. This symbol appears as a small state with an ellipsis label, which signifies the existence of one or more states that the view is suppressing, The textual equivalence of the state suppression symbol shall be the reserved phrase "or other states".

EXAMPLE

A can be **s1, s2, s3, s4,** or **s5.**
P changes **A** from **s1** to **s3.**

A can be **s1, s3,** or other states**.**
P changes **A** from **s1** to **s3.**

Figure 45 — A stateful object with all states expressed (left) and a suppressed version (right)

14.2.1.2 Unfolding and folding

Unfolding shall be a mechanism for refinement, elaboration, or decomposition. Unfolding shall reveal a set of things that relate to the unfolded thing. The result of unfolding shall be a hierarchy tree, the root of which shall be the unfolded thing. Linked to the root shall be the things that constitute the elaboration of the unfolded thing.

Conversely, folding shall be a mechanism for abstraction or composition, which shall apply to an unfolded hierarchical tree. Folding shall hide the set of unfolded things, leaving just the root.

Each of the four fundamental structural relation links may apply unfolding and folding.. The four kinds of unfolding-folding pairs shall be:

- aggregation unfolding—exposing the parts of a whole, and participation folding—hiding the parts of a whole;
- exhibition unfolding—exposing the exhibitor's features, and characterization folding—hiding the exhibitor's features;
- generalization unfolding—exposing the specializations of the general, and specialization folding—hiding the general's specializations; and
- classification unfolding—exposing the class instances, and instantiation folding—hiding the class instances

In-diagram unfolding shall occur when the refineable and its refinees appear unfolded in the same OPD. Because unfolding uses the fundamental structural links, in-diagram unfolding is graphically, syntactically and semantically equivalent to using fundamental structural links.

New-diagram unfolding shall occur when the refineable and its refinees appear unfolded in a new OPD.

Graphically, the refineable shall have a thick contour in both the more abstract OPD in which the refineable appears folded without refinees, and in the new more detailed OPD context, in which the refineable appears unfolded and connects to its refinees with one or more fundamental structural link.

The corresponding OPL sentence for the new-diagram OPD where the refineable has n refinees shall be: **Refineable** unfolds into **Refinee**$_1$, **Refine**$_2$,…, and **Refine**$_n$

NOTE 1 Unfolding may be more precisely specified as part-unfolding, feature-unfolding, specialization-unfolding, and instance-unfolding (see A.4.7.2).

The modeller decision whether to use in-diagram or new-diagram unfolding should account for the trade-off between the clutter added to the current OPD and the need to create a new OPD for displaying the refinees and associated links amongst them.

NOTE 2 Unfolding often occurs as a combination of new-diagram and in-diagram unfolding to represent multiple elaboration or decomposition situations.

NOTE 3 Partial unfolding may be depicted in the same manner as a partial fundamental structural relation link.

To satisfy a particular contextual relevance for an OPD, a modeller may choose which refinees appear unfolded. Following the bimodal representation of OPM, the OPL corresponding to the OPD shall express only those refinees that appear in that OPD.

NOTE 4 Partial folding is equivalent to partial unfolding where the collections of each are complementary.

NOTE 5 Unfolding reveals finer structural details rather than behaviour, i.e. no transfer of execution control occurs, see 14.2.2. However, hierarchical dependencies involving procedural links may result in behavioural changes associated with use of the unfolded thing.

14.2.1.3 In-zooming and out-zooming

In-zooming shall be a kind of unfolding that combines aggregation-participation and exhibition-characterization with additional semantics. For processes, in-zooming enables modelling the subprocesses, their temporal order, their interactions with objects, and passing of execution control to and from that context. For objects, in-zooming creates a distinct context that enables modelling of the constituent objects' spatial or logical order.

Graphically, for both in-diagram and new-diagram process in-zooming, the ellipse of the refineable enlarges to accommodate the symbols for the refinees, and the links amongst them, which are within the in-zoom context. In the case of new-diagram in-zooming, the refineable shall have a thick contour in both the more abstract OPD in which the refinealbe appears without refinees, and in the new more detailed OPD context, in which the refineable appears surrounding the subprocess refinees and attendant objects..

The corresponding process in-zoom OPL sentence shall be: **Process** zooms into **Subprocess A, Subprocess B,** and **Subprocess C,** in that sequence

NOTE 1 In zooming may be more precisely specified by indicating the abstract OPD name and the more detailed OPD name (see A.4.7.4).

The context of an in-zoomed process shall include the subprocesses, which are parts of the in-zoomed process, and possibly interim objects that are attributes of the in-zoomed process. The contextual scope of the in-zoomed process shall be the refineable, its subprocesses, attributes and links as depicted in the OPD.

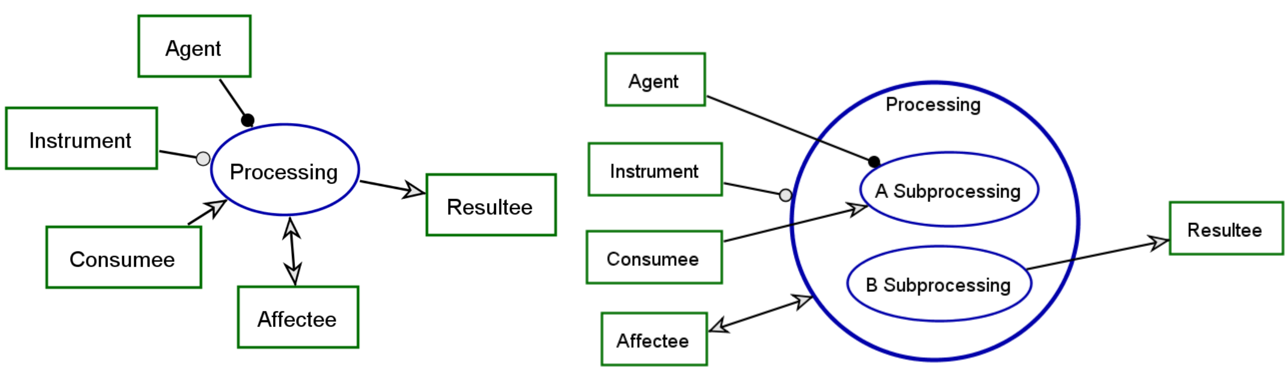
The execution timeline within the context of an in-zoomed process shall flow from the top of its enlarged process ellipse symbol to the bottom of that ellipse. This timeline shall depict the sequence of subprocess invocations. The vertical arrangement of the top point of the subprocess ellipse symbols within the outer process ellipse shall indicate the nominal execution sequence of the subprocesses within the context of the process.

Analogous to process in-zooming, object in-zooming shall expose constituent objects as parts of the in-zoomed object and possibly interim processes that are in-zoomed object operations within the scope of the in-zoomed object context. Unlike in-zooming a process, in-zooming an object does not result in a transfer of execution control. The consequence of new-diagram object in-zooming is a context shift from the object as part of a larger OPD context to the object as the entire OPD context in which the constituent parts of the object are exposed and spatially or logically ordered.

Graphically, the rectangle of the in-zoomed object enlarges to accommodate the symbols for the refinees, and the links amongst them. The arrangement of the object rectangles within the context of the in-zoomed object enlarged rectangle shall indicate spatial arrangement or logical order of the objects. This enables ordered enumeration of data, such as in a vector or a matrix.

The corresponding object in-zoom OPL sentence shall be: **Object** zooms into **Subobject A, Subobject B,** and **Subobject C,** in that sequence**.**

EXAMPLE 1 Figure 46 depicts abstract Processing in SD, the System Diagram, and details of Processing in SD1 after zooming into Processing, showing its two subprocesses.

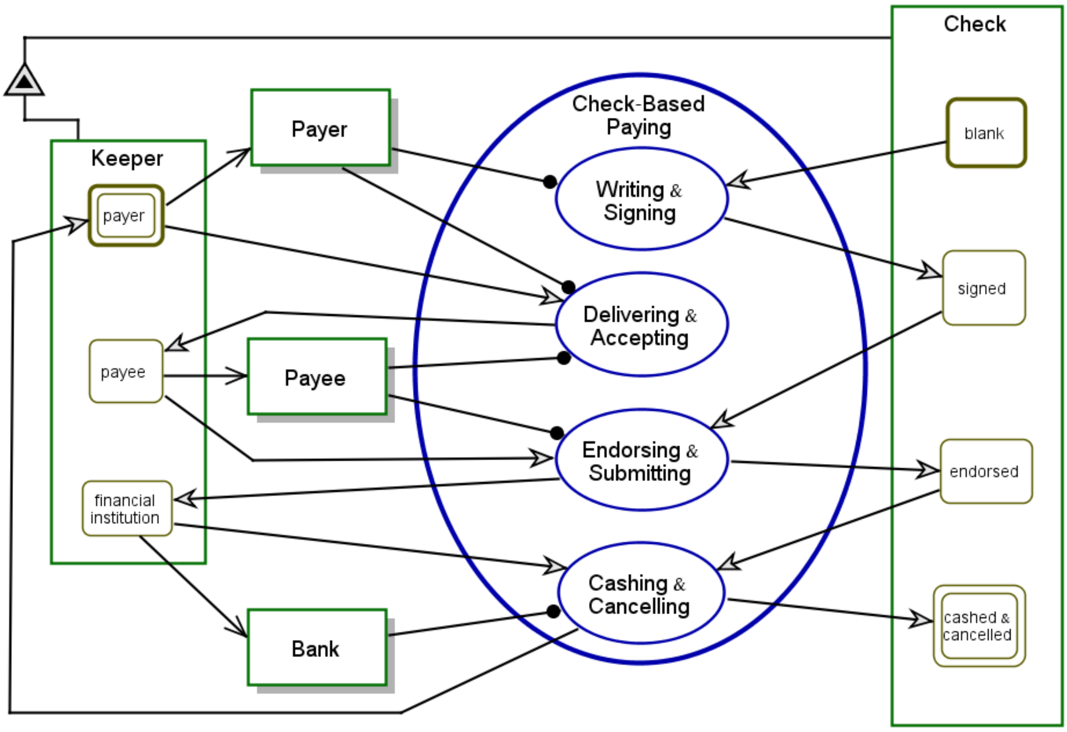


Agent handles **Processing.**
Processing requires **Instrument.**
Processing consumes **Consumee.**
Processing affects **Affectee.**
Processing yields **Resultee.**

Processing requires **Instrument.**
Processing affects **Affectee.**
Processing zooms into **A Subprocessing** and **B Subprocessing** in that sequence.
Agent handles **A Subprocessing.**
A Subprocessing consumes **Consumee.**
B Subprocessing yields **Resultee.**

Figure 46 — New-diagram in-zooming generic example

EXAMPLE 2 Figure 47 depicts the Check-Based Paying process of Figure 29 with in-zooming to expose the sequence of subprocesses and the allocation of links from the process to its subprocesses.



Check exhibits **Keeper.**
Check can be **blank, signed, endorsed,** or **cashed & cancelled.**
State **blank** of **Check** is **initial.**
State **cashed & cancelled** of **Check** is **final.**
Keeper can be **payer, payee,** or **financial institution.**
State **payer** of **Keeper** is **initial** and **final.**
Payer Keeper relates to **Payer.**
Payee Keeper relates to **Payee.**
Financial institution Keeper relates to **Bank.**
Check-Based Paying zooms into **Writing & Signing, Delivering & Accepting, Endorsing & Submitting,** and **Cashing & Cancelling in that sequence.**
Payer handles **Writing & Signing** and **Delivering & Accepting.**
Payee handles **Delivering & Accepting** and **Endorsing & Submitting.**
Bank handles **Cashing & Cancelling.**
Writing & Signing changes **Check** from **blank** to **signed.**
Delivering & Accepting changes **Keeper** from **payer** to **payee.**
Endorsing & Submitting changes **Check** from **signed** to **endorsed.**
Cashing & Cancelling changes **Check** from **endorsed** to **cashed & cancelled** and **Keeper** from **bank** to **payer.**

Figure 47 — Check-Based Paying process with in-zooming to expose its four sequential subprocesses

NOTE 2 In-zooming expresses process behaviour that is the result of structural links and procedural links indicating a dynamic transfer of execution control among OPD models. The operational execution context shifts from the process to the in-zoomed OPD and then back to the process.

14.2.2 Control (operational) semantics within an in-zoomed process context

14.2.2.1 Implicit invocation link

In-zooming a process shall specify a transfer of execution control to subprocesses at a different extent of detail. Executing a process with an in-zoomed context shall recursively transfer execution control to the top-most subprocess(es) within that process context, which is in a different OPD in case of new-diagram in-zooming. Execution control shall return to the in-zoomed process after its final enabled subprocess completes.

The implicit invocation link shall be a set of invocation links between a process and an in-zoom subprocess, two subprocesses within the context of an in-zoomed process, or an in-zoomed subprocess and its process. Similar to its explicit counterpart, the implicit invocation link shall signify the invocation of a subsequent process or concurrently beginning processes.

Upon arriving at an in-zoomed process context, execution control shall immediately transfer to the subprocess(es) with the highest ellipse (oval) top-most point within this process in-zoom context. The implicit invocation link from a process to its top-most in-zoom subprocess transfers execution control. Along the process timeline, the completion of a source subprocess immediately invokes the subsequent subprocess(es) using the implicit invocation link. Upon completion of the subprocess with an ellipse top-most point that is lowest within this in-zoom context, execution control shall return to the in-zoomed process along the implicit invocation link.

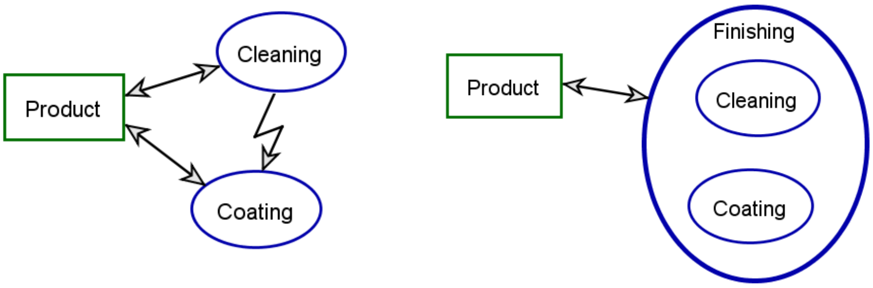
Since invocation is an event, satisfaction of the precondition for each subprocess is necessary to allow that subprocess to perform.

When two or more subprocesses have their top-most ellipse points at the same height, then an implicit invocation link shall initiate each process and they shall start in parallel upon individual precondition satisfaction. The process that completes last shall initiate the next process or set of parallel subprocesses.

Graphically, no symbol explicitly denotes the implicit invocation link. The top-to-bottom vertical arrangement of the top-most point of the subprocess ellipse symbols within the context of the in-zoomed process shall denote an implicit invocation link between successive subprocesses in that arrangement.

The syntax of an implicit invocation link OPL sentence shall be: **Process** zooms into **Subprocess A** and **Subprocess B**, in that sequence.

EXAMPLE In the OPD on the left hand side of Figure 48, **Cleaning** invokes **Coating**, so **Cleaning** affects **Product** first and then **Coating** affects **Product**. The invocation link dictates this process sequence. In the equivalent OPD on the right hand side of Figure 48, **Finishing** zooms into **Cleaning** and **Coating**, with the former's ellipse top point above the latter's, so when **Finishing** starts, execution control immediately transfers to **Cleaning**, and when **Cleaning** ends, the implicit invocation link invokes **Coating**. The two OPDs are semantically equivalent, except that the one on the left does not have **Finishing** as an enclosing context, making it less expressive from a system viewpoint while using more graphical elements.



Cleaning affects **Product.**
Cleaning invokes **Coating.**
Coating affects **Product.**

Finishing affects **Product.**
Finishing zooms into **Cleaning** and **Coating**, in that sequence**.**

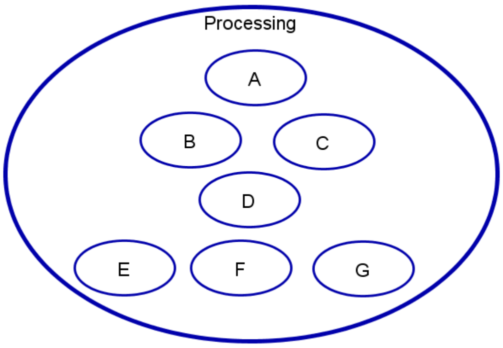
Figure 48 — Invocation link (left) and implicit invocation link (right)

14.2.2.2 Implicit parallel invocation link set

Graphically, when the ellipse top points of two or more subprocesses within the scope of an in-zoomed process are at the same height (with possible allowable tolerance), these subprocesses shall begin in parallel, subject to precondition satisfaction for both. In this situation, there is a set of implicit invocation links from the source process of the implicit invocation link to each one of the parallel processes.

The heights of the enclosed subprocesses' ellipse top points induce a partial order among these subprocesses. Subprocesses whose ellipse top points are at the same height start in parallel. When the last one of these subprocesses ends, i.e. process synchronization occurs, execution control shall attempt to invoke the next subprocess. If there are two or more subprocesses with a lower ellipse-top point at the same height, the execution control invokes them in parallel. If there are no more subprocesses to invoke, execution control returns to the in-zoomed refineable process.

The syntax of the implicit parallel invocation link OPL sentence shall be: **Process** zooms into parallel **Subprocess A** and **Subprocess B.**



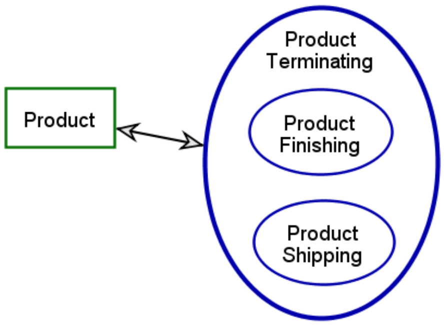
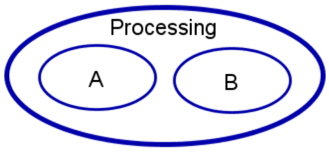
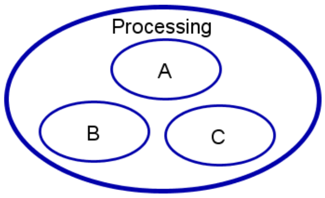
Processing zooms into **A,** parallel **B** and **C, D,** and parallel **E, F, G,** in that sequence**.**

Figure 49 — Partial subprocesses order and implicit parallel invocation link set

EXAMPLE Figure 49 shows subprocesses with the following partial order: **A**, (**B**, **C**), **D**, (**E**, **F**, **G**). **B** and **C** start upon completion of **A**. **D** starts upon completion of the longer process from among **B** and **C**. **E**, **F**, and **G** start upon completion of **D**. Execution control returns to Processing upon completion of the longer process from among **E**, **F**, and **G**.

14.2.2.3 Implicit invocation link summary

Table 24 — Implicit invocation link summary

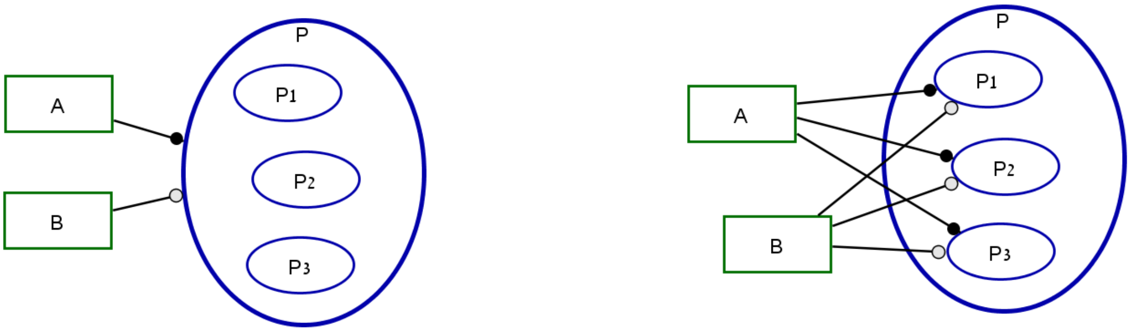
Name	Semantics	Sample OPD & OPL	Source	Destination
Implicit invocation link	Upon subprocess completion within the context of an in-zoomed process, the subprocess immediately invokes the one(s) below it.	 **Product Terminating** zooms into **Product Finishing** and **Product Shipping,** in that sequence**.**	Initiating process, whose ellipse top point is above the initiated process	Initiated process, whose ellipse top point is below the ellipse top point of the initiating process
Parallel Implicit invocation link set	Top: Subprocesses A and B initiate in parallel as soon as Processing starts. Bottom: Subprocesses B and C initiate in parallel as soon as subprocess A ends.	 **Processing** zooms into **parallel A** and **B.**  **Processing** zooms into **A** and parallel **B** and **C,** in that sequence**.**	Initiating process, whose ellipse top point is above the set of initiated processes, whose ellipse top points are at the same height (within a pre-determined tolerance).	A set of initiated processes, whose ellipse top points are at the same height (within tolerance) and below the initiating process ellipse top point

14.2.2.4 Link distribution across context

14.2.2.4.1 Semantics of link distribution

Graphically, a procedural link attached to the contour of an in-zoomed process has distributive semantics. Leaving a link attached to the contour of the in-zoomed process shall mean that the link is distributed and attached to each one of the subprocesses. The contour of the in-zoomed process has semantics analogous to that of algebraic parentheses following a multiplication symbol, which distribute the multiplication operator to the expressions inside the parentheses.

EXAMPLE 1 In Figure 50, the OPDs on the left and right are equivalent, but the one on the left is clearer and less cluttered. An agent link from **A** to **P** means that **A** handles the subprocesses **P1**, **P2**, and **P3**. An instrument link from **B** to **P** means that the subprocesses **P1**, **P2**, and **P3** require **B**. Analogously in algebra, suppose the agent (or instrument) link was a multiplication operator, A was a multiplier and in-zooming was addition, such that **P** = **P1** + **P2** + **P3**, and **P** was a multiplicand, then **A*****P** = **A***(**P1** + **P2** + **P3**) = **A*****P1** + **A*****P2** + **A*****P3**.



A handles **P.**
P requires **B.**
P zooms into **P1, P2,** and **P3,** in that sequence**.**

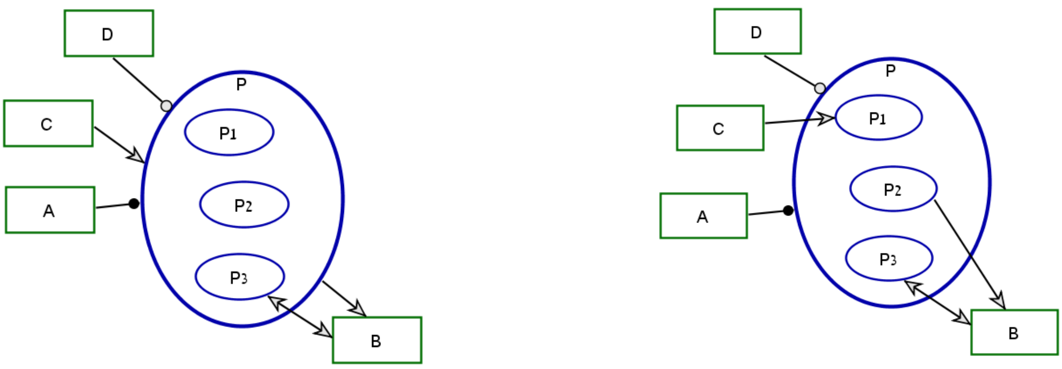
P zooms into **P1, P2,** and **P3,** in that sequence**.**
A handles **P1, P2,** and **P3.**
P1, P2, and **P3** require **B**

Figure 50 — In-zooming link distribution

If an enabler connects to the outer contour of an in-zoomed contour it shall connect to at least one of its subprocesses. Consumption and result links shall not be attached to the outer contour of an in-zoomed process because this violates temporal logical conditions. With a distributed consumption link, an attempt would be made to consume an already-consumed object by a subprocesses that is not the first to perform. Similarly, a distributed result link would attempt to create an already existing object instance.

NOTE 1 The modeller needs to be careful when more than one process creates the same object, i.e. more than one operational instance of the object exists, or more than one process affect or consume the same object. OPM modelling tools need to track the number of operational instances of an object.

EXAMPLE 2 In Figure 51 the OPD on the left contains invalid consumption and result links, as annotated in the OPL**.** The consumption link gives rise to the OPL sentence "**P** consumes **C**." Applying link distribution, the consequence is the three OPL sentences "**P1** consumes **C**.", "**P2** consumes **C**.", and "**P3** consumes **C**.". However, since **P1** consumes **C** first according to its temporal order, the same instance of **C** does not exist when **P2** or **P3** performs and therefore **P2** and **P3** cannot consume **C** again. Similarly, the same operational instance of **B** results only once. The OPD on the right depicts validity links by specifying which of the subprocesses of **P** consumes **C** (**P1**) and which one yields **B** (**P2**).



A handles **P.**
P requires **D.**
P zooms into **P1, P2,** and **P3,** in that sequence**.**
P consumes **C.** – NOT VALID!
P yields **B.** – NOT VALID!
P3 affects **B.**

A handles **P.**
P requires **D.**
P zooms into **P1, P2,** and **P3,** in that sequence**.**
P1 consumes **C.**
P2 yields **B.**
P3 affects **B.**

Figure 51 — Link distribution restriction for consumption and result links

Since attaching a consumption or result link to an in-zoomed process is invalid, when a process is in-zoomed, all the consumption and result links that were attached to it shall be attached initially or by default to its first subprocess.

NOTE 2 A modelling tool should automatically establish default semantics, which the modeller may modify.

EXAMPLE 3 In Figure 51 as soon as the modeller in-zooms **P** and inserts **P1** into its context, the destination end of the consumption link from **C** migrates from **P** to **P1**. Similarly, the source end of the result link to **B** also migrates from **P** to **P1**. When the modeller adds **P2**, the modeller may migrate the destination end of the consumption link and/or the source end of the result link from **P1** to **P2**, as Figure 51 shows.

14.2.2.4.2 Event link constraint

An event link shall not cross the boundary of an in-zoomed process from the outside of that process to initiate any one of its subprocesses at any level, because this amounts to an attempt to interfere with the prescribed temporal order of the synchronous in-zoomed process.

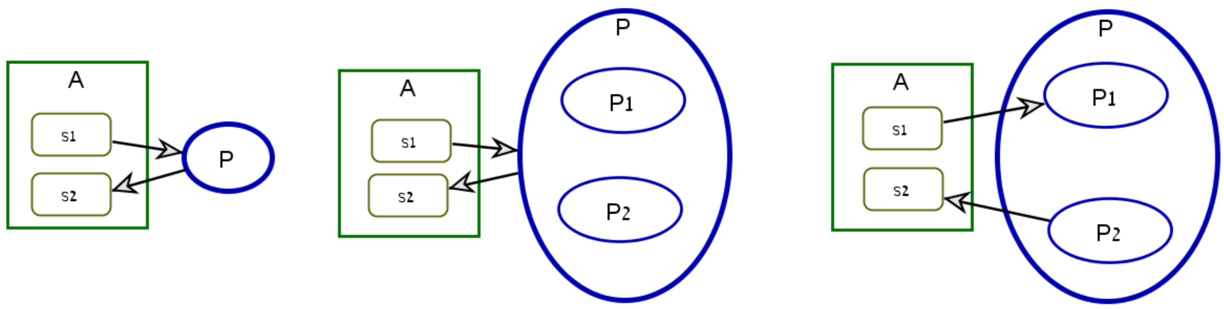
If the skipped process is within an in-zoom context and there is a subsequent process in this context, execution control initiates that process, otherwise execution control transfers back to the in-zoomed process.

14.2.2.4.3 Split state-specified transforming links

When a process that changes an object from an input state to an output state is in-zoomed, the OPD, either in-diagram or new-diagram, becomes underspecified. To restore specification, the modeller shall attach both the state-specified input link and the state-specified output link to one of the subprocesses in a temporally-feasible manner. Splitting the input-output specified link pair in two shall signify the split state-specified transforming link pair.

Graphically, two links to an object with two or more states connecting across a process contour to different subprocesses with one state-specified input link and one state-specified output link shall denote the split state-specified transforming link.

EXAMPLE 1 In Figure 52 the OPD in the middle is underspecified because **P1** or **P2** could each change **A** from **s1** to **s2**, or **P1** could change **A** from **s1** and **P2** could change **A** to **s2**. The OPD on the right models this last case, giving rise to a new split input link from **s1** of **A** to **P1** and a new split output link from **P2** to **s2**.



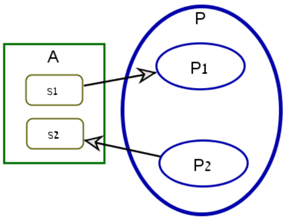
A can be **s1** or **s2.**
P changes **A** from **s1** to **s2.**

A can be **s1** or **s2.**
P zooms into **P1** and **P2,**
in that sequence**.**
P changes **A** from **s1** to **s2.**
– *UNDERSPECIFIED!*

A can be **s1** or **s2.**
P zooms into **P1 and P2,**
in that sequence**.**
P1 changes **A** from **s1.**
P2 changes **A** to **s2.**

Figure 52 — Split state-specified transforming link to resolve under specification

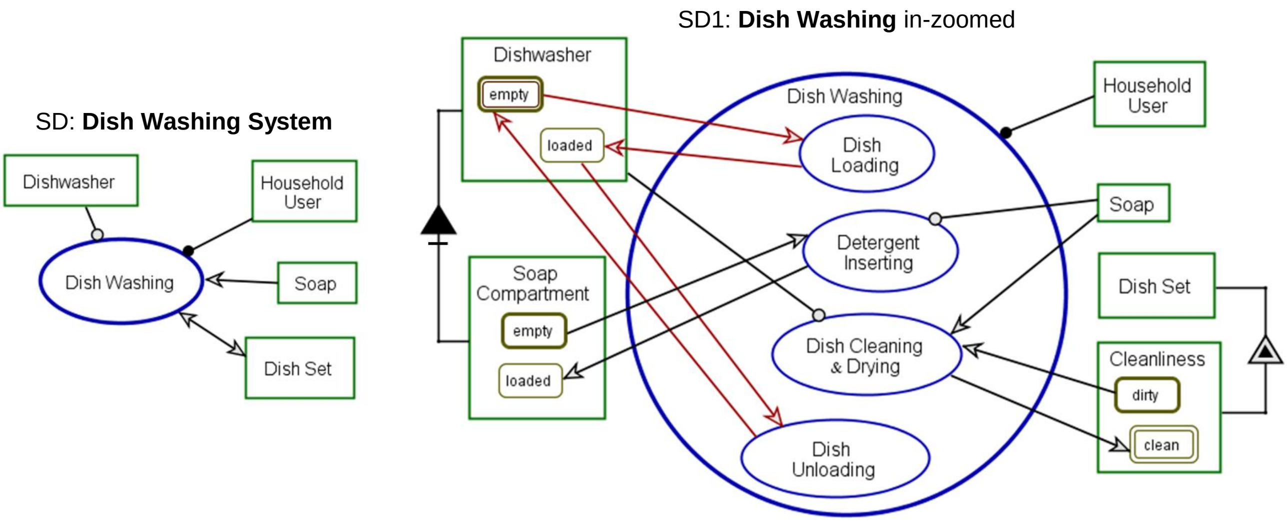
Table 25 - Split input-output specified effect link pair

Name	Semantics	Sample OPD & OPL	Source	Destination
Split input-output specified effect link pair *The top arrow:* split input-specified effect link *The bottom arrow:* split output-specified effect link	An early subprocess of an in-zoomed process takes an object out of its input state. A late subprocess of the same in-zoomed process changes the object to be in its output state.	 **P1** changes **A** from **s1.** **P2** changes **A** to **s2.**	*The top arrow:* Input state of an affected object *The bottom arrow:* Late subprocess of an in-zoomed process	*The top arrow:* Early subprocess of an in-zoomed process *The bottom arrow:* Output state of the affected object

NOTE 1 There are no control-modified versions of the split input-specified effect link.

NOTE 2 An object may have the role of an instrument in an abstract OPD and a transformee in another descendent, more detailed and concrete OPD. At the abstract OPD, the process does not appear to affect the object, because the object's initial state is the same as its final state. Therefore, at the abstract OPD the object is an instrument, as indicated by an instrument link. However, at a descendent, more concrete OPD, that same process does appear to change the state of that object from the initial state and then back to the initial state.

EXAMPLE 2 In **Figure 53** the left System Diagram (SD: **Dish Washing System**), a **Dishwasher** object is an instrument to **Dish Washing** process, since no change in state of the **Dishwasher** is visible at that extent of abstraction. In the descendent OPD (SD1: **Dish Washing** in-zoomed), **Dish Washing** zooms into **Loading** (of a **dirty Dish Set**), **Cleaning** (which changes **Dish Set** from **dirty** to **clean**), and **Unloading** (of a **clean Dish Set**). **Loading** changes the state of **Dishwasher** from **empty** to **loaded**, while **Unloading** changes it back from **loaded** to **empty**, so **empty** is both the initial and final state. While the **Dishwasher** is an instrument in the System Diagram, at the more detailed descendent OPD, the **Dishwasher** is an affectee—it becomes **loaded** and then **empty** again. The only effect visible in the System Diagram is the effect on **Dish Set**.



Household User handles **Dish Washing.**
Dish Washing requires **Dishwasher.**
Dish Washing consumes **Soap.**
Dish Washing affects **Dish Set.**

Dish Washer consists of **Soap Compartment** and other parts**.**
Dishwasher can be **empty** or **loaded.**
State **empty** of **Dishwasher** is initial and final**.**
Soap Compartment can be **empty** or **loaded.**
State **empty** of **Soap Compartment** is initial**.**
Dish Set exhibits **Cleanliness.**
Cleanliness of **Dish Set** can be **dirty** or **clean.**
State **dirty** of **Cleanliness** of **Dish Set** is initial**.**
State **clean** of **Cleanliness** of **Dish Set** is final**.**
Household User handles **Dish Washing.**
Dish Washing zooms into **Dish Loading, Detergent Inserting, Dish Cleaning & Drying,** and **Dish Unloading,** in that sequence**.**
Dish Loading changes **Dishwasher** from **empty** to **loaded.**
Detergent Inserting requires **Soap.**
Detergent Inserting changes **Soap Compartment** from **empty** to **loaded.**
Dish Cleaning & Drying requires **Dishwasher.**
Dish Cleaning & Drying consumes **Soap.**
Dish Cleaning & Drying changes **Cleanliness** of **Dish Set** from **dirty** to **clean.**
Dish Unloading changes **Dishwasher** from **loaded** to **empty.**

Figure 53 — Role of abstraction with split state transforming links

14.2.2.4.4 Operational instances of involved object set

As a consequence of link distribution, the following constraints shall apply to operational instances of transformees:

— each consumee operational instance in the preprocess object set of a process shall cease to exist at the beginning of the most detailed subprocess of that process, which consumes the operational instance, and the operational instance is not in the postprocess object set of that process**;**

— each affectee operational instance in the preprocess object set of a process that changes that operational instance as a consequence of the process performance shall exit from its input state, the state from which it changes, at the beginning of the most detailed subprocess that changes the affectee;

— each affectee operational instance in the postprocess object set of a process that changes that operational instance as a consequence of the process performance shall enter its output state at the completion of the most detailed subprocess that changes the affectee; and,

— each resultee operational instance in the postprocess object set of a process shall begin existence at the completion of the most detailed subprocess that yields the resultee operational instance and the operational instance is not in the preprocess object set of that process.

NOTE 1 A stateful object B for which the execution of process P has the effect of changing the state of B, exits from the input state at the beginning of the most detailed subprocess of P that changes B, and enters the output state at the end of the same subprocess of P or some subsequent subprocess of P. Since process P execution takes a positive amount of time, that object B is in transition between states, from its input state to its output state: it has left its input state but has not yet arrived at its output state.

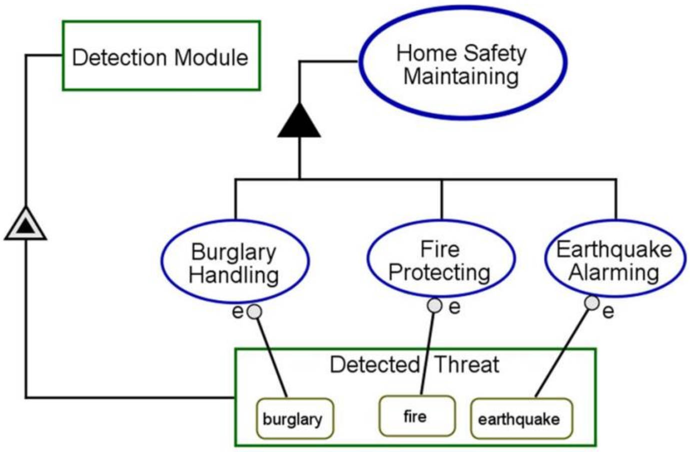
14.2.2.5 Synchronous vs. asynchronous process refinement

Since the aggregation-participation fundamental structural relation does not prescribe any "partial order" of process performance, the modelling of synchronous process refinement shall use in-zooming.

EXAMPLE 1 The system in Figure 53 is synchronous: there is a fixed, well-defined order of each subprocess within the in-zoom context of **Dish Washing.**

The modelling of asynchronous process refinement shall use the aggregation-participation fundamental structural link either through in-diagram aggregation unfolding or as a new-diagram aggregation unfolding of the process.

EXAMPLE 2 **Figure 54** depicts a portion of a Home Safety System that carries out the function **Home Safety Maintaining**, which includes the subprocesses **Burglary Handling**, **Fire Protecting**, and **Earthquake Alarming**. Since the order of these three subprocesses is unknown, the OPD uses in-diagram aggregation unfolding with an aggregation-participation link from this function rather than an in-zoomed version of **Home Safety Maintaining**. Home Safety Maintaining in-zooms to a recurring systemic process, Monitoring & Detecting, for which Detection Module is an instrument and Threat Appearing is an environmental process.



Home Safety Maintaining consists of **Burglary Handling**, **Fire Protecting**, and **Earthquake Alarming.**
Detection Module exhibits **Detection Treat.**
Detection Treat can be **burglary**, **fire**, or **earthquake.**
Burglary Detected Threat initiates **Burglary Handling**, which requires **burglary Detected Threat**.
Fire Detected Threat initiates **Fire Protecting**, which requires **fire Detected Threat**.
Earthquake Detected Threat initiates **Earthquake Alarming**, which requires **earthquake Detected Threat**

Figure 54 — Home Safety Maintaining is an asynchronous system

14.2.2.6 Expressing the contextual texture of a system

14.2.2.6.1 Navigating the contexts of a system

14.2.2.6.1.1 The OPD process tree

An OPD process tree, also called OPD tree, shall be a directed tree graph with root of SD, the System Diagram, and the other OPDs as nodes with their OPD labels. The directed edges of an OPD tree shall have labels with each edge pointing from the parent OPD, which contains the refineable element, to a child OPD containing refinees, which elaborates a process in the parent OPD via new-diagram in-zooming for synchronous subprocesses or new-diagram aggregation unfolding for asynchronous subprocesses.

14.2.2.6.1.2 The OPD object tree

Unlike the OPD process tree that has a single root, the OPD object tree is more like a forest of many trees, each stemming from a distinct refineable object that unfolds or in-zooms to reveal detail. Rather than identifying the possible flow of execution control found in the OPD process tree, the OPD object tree shall encapsulate the information about an object as a hierarchic structure. The system execution should maintain dependencies among OPD object tree elements and between OPD object trees.

NOTE OPM tools provide rules for model construction that enforce the maintenance of dependencies during model creation.

14.2.2.6.1.3 OPM diagram labels

The OPM system name shall be the name of the OPM model that specifies the system. An OPD name is the name that identifies each OPD in the OPD process tree.

SD shall be the label designation for the root OPD in the OPD tree hierarchy. This SD occupies tier 0 in the OPD hierarchy tree and shall have exactly one OPD; higher numbered tiers, i.e. those corresponding to successive refinements, may have one or more OPDs. SD shall contain one and only one systemic process, which represents the overarching system function that delivers functional value to stakeholders. SD may contain one or more environmental processes.

14.2.2.6.1.4 OPD process tree edge label

Each edge in the OPD process tree shall have a label. The label shall express a refinement relation that corresponds to the implicit invocation link or unfolding relation. Considering each OPD to be an object and the entire OPD process tree to be a single OPD, each edge shall be a unidirectional tagged structural link with a tag of "is refined by in-zooming <**Refineable Name**> in ", or "is refined by unfolding <**Refineable Name**> in ".

An OPD refinement OPL sentence shall be an OPL sentence describing the refinement relation between a refineable present in a $tier_N$ OPD and the $tier_{N+1}$ refinement OPD.

The syntax of an in-zoomed OPD refinement OPL sentence shall be: "<**Tier$_N$ OPD label**> is refined by in-zooming <**Refineable Process Name**> in "<**Tier$_{N+1}$ OPD Label**>."

The syntax of an unfolded OPD refinement OPL sentence shall be: "<**Tier$_N$ OPD label**> is refined by unfolding <**Refineable Process Name**> in "<**Tier$_{N+1}$ OPD Label**>."

14.2.2.6.1.5 System map and model views

A system map shall be an OPD process tree that explicitly depicts the element (things and links) content of each OPD (node). Because the system map may become very large and unwieldy, mechanisms shall allow access to model content and the associations among elements. These mechanisms, collectively referred to as model views consisting of model facts, shall include a list of all things and the OPDs in which they appear, the OPD process tree, and the OPD object trees.

In addition, an OPM tool set should provide a mechanism for creating views, as OPDs with associated OPL sentences, of objects and processes that meet specific criteria. These views may include the critical path for minimal system execution duration, or a list of system agents and instruments, or an OPD of objects and processes involved in a specific kind of link or set of links.

EXAMPLE An OPD can be created by (1) refining (unfolding or in-zooming) an object or (2) collecting and presenting in a new OPD things that appear in various OPDs for expressing assignment of system sub-functions to system-module objects.

14.2.2.6.2 Whole System OPL specification

An OPL paragraph shall be the collection of OPL sentences that together specify in text the semantic expression of the corresponding OPD.

NOTE 1 An OPL paragraph name, using the OPD name, may precede the first OPL sentence of each OPL paragraph.

An OPM system model shall be the collection of successive OPL paragraphs corresponding to the collection of OPDs present.

An entire OPL specification of a system should begin with an OPL specification starting title. The OPL paragraphs follow the title in successive blocks, each beginning on a new line with the corresponding OPD and the OPL paragraph sentences following.

NOTE 2 The sequence of OPL paragraphs should begin with the SD and generally follow breadth-first, unless the modeller identifies a different sequence.

EXAMPLE Table 26 contains the entire OPL specification of the OPM model in Figure 53.

Table 26 — Whole system OPL for Dish Washing System

<u>OPL specification of **Dish Washing System**</u>

<u>SD: **Dish Washing System**</u>
Household User handles **Dish Washing.**
Dish Washing requires **Dishwasher.**
Dish Washing consumes **Soap.**
Dish Washing affects **Dish Set.**

<u>SD is refined by in-zooming **Dish Washing** in SD1</u>**.**

<u>SD1: **Dish Washing** in-zoomed</u>
Dish Washer consists of **Soap Compartment** and other parts**.**
Dishwasher can be **empty** or **loaded.**
State **empty** of **Dishwasher** is initial and final**.**
Soap Compartment can be **empty** or **loaded.**
State **empty** of **Soap Compartment** is initial**.**
Dish Set exhibits **Cleanliness.**
Cleanliness of **Dish Set** can be **dirty** or **clean.**
State **dirty** of **Cleanliness** of **Dish Set** is initial**.**
State **clean** of **Cleanliness** of **Dish Set** is final**.**
Household User handles **Dish Washing.**
Dish Washing zooms into **Dish Loading, Detergent Inserting, Dish Cleaning & Drying,** and **Dish Unloading,** in that sequence**.**
Dish Loading changes **Dishwasher** from **empty** to **loaded.**
Detergent Inserting requires **Soap.**
Detergent Inserting changes **Soap Compartment** from **empty** to **loaded.**
Dish Cleaning & Drying requires **Dishwasher.**
Dish Cleaning & Drying consumes **Soap.**
Dish Cleaning & Drying changes **Cleanliness** of **Dish Set** from **dirty** to **clean.**
Dish Unloading changes **Dishwasher** from **loaded** to **empty.**

<u>End of OPL specification of **Dish Washing System**</u>

14.2.3 OPM fact consistency principle

The fact consistency OPM principle stipulates that:

(1) a model fact appearing in one OPD shall be true for the entire collection of OPDs within the OPM system model, and

(2) no OPD in the OPD process tree or an OPD object tree shall contain a model fact that contradicts a model fact in the same OPD or in another OPD.

A fact in one OPD may be a refinement or an abstraction of a fact in a different OPD within the same OPM system model.

NOTE This principle does not preclude the possibility of representing any model element any number of times in as many OPDs as the modeller wishes. Since a link cannot exist without the things it links, if a link is present then the two things on its ends need to be present as well.

EXAMPLE It is not possible for one OPD to express the fact that "**P** yields **A.**" and for the same or another OPD in the same OPD tree to express the fact that "**P** consumes **A.**" However, it is permissible for one OPD to express the fact that "**P** affects **A.**" and for another OPD in the same OPD tree to express the fact that "**P** changes **A** from **s1** to **s2.**" because the latter fact is a refinement, not a contradiction of the former.

14.2.4 Abstraction ambiguity resolution for procedural links

14.2.4.1 Abstraction and procedural link precedence

Out-zooming abstracts a collection of related things, the refinees and associated links, into a refineable. When the modeller performs the abstraction, the procedural links between refinees and things that are not refinees, shall migrate to the context of the OPD that depicts the refineable. This migration may cause a situation in which two or more procedural links of different kinds link an object and a process. According to the procedural link uniqueness OPM principle (see 8.1.2) an object or an object state shall link to a process by only one procedural link. To sustain this principle, the modeller shall resolve the conflict between candidate links to determine which remains or which new link replaces the candidates in the abstract OPD. The loss of detail information is consistent with the notion of abstraction.

EXAMPLE Figure 55 demonstrates the problem of procedural link abstraction. In SD1, the result link from **P1** to **B** is more significant than the effect link from **P2** to **B**, so when **SD1** is out-zoomed to **SD**, the result link prevails.

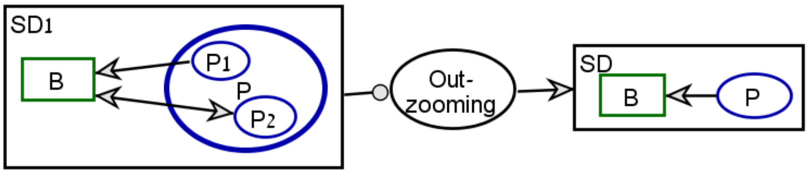


Figure 55 — Abstracting procedural links

Semantic strength and link precedence are two concepts to guide the determination of which links to retain and which to hide when an OPD is out-zoomed or folded.

Semantic strength of a procedural link shall be the significance of the information that the link carries. Information concerning a change in existence, either creation or elimination, is more significant than information about change to an existing thing. The relative semantic strength of the two conflicting procedural links shall determine link precedence. When two or more procedural links compete to remain represented in an OPD abstraction of refinement, the link that prevails is the one with the highest semantic strength.

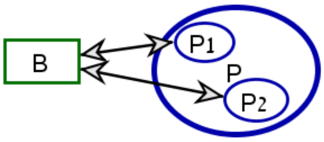
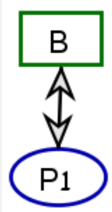
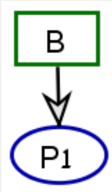
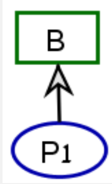
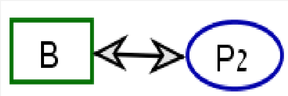
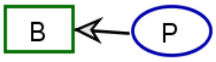
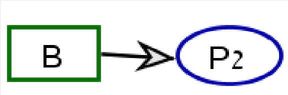
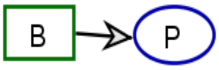
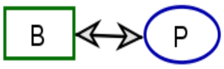
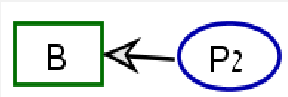
NOTE The concept of link precedence allows the modeller to resolve conflicts in representation amongst OPD contexts and guides the modeller in establishing appropriate procedural links at the various extents of detail.

14.2.4.1.1 Precedence among transforming links

Transforming links include result, effect, and consumption links. Since object creation and consumption are semantically stronger, i.e. they have higher semantic strength than affecting the object by changing its state, result and consumption links have precedence over effect links, as demonstrated in Figure 55. However, since result and consumption links are semantically equivalent, when they compete, the prevailing link shall be the effect link because the effect link allows both creation and elimination as effects.

Table 27 shows transforming link precedence: **P** in the upper left corner is out-zoomed. The column headings show the three possible transforming links between **P1** and **B**, while the row headings show the three possible links between **P2** and **B**. The table cells show the prevailing link between **B** and **P** after **P** is out-zoomed. Cells marked as "Invalid" indicate the impossibility of the combination. For example, inspecting the centre cell, if **P1** consumes **B**, **B** no longer exists when **P2** later tries to consume it again.

Table 27 – Transforming link precedence

Zoomed-out process P: 			
		Invalid	
		Invalid	
	Invalid		Invalid

14.2.4.1.2 Precedence among transforming and enabling links

Transforming links are semantically stronger than enabling links, because transforming links denote creation, consumption, or change of the linked object, while the enabling links only denote enablement. A transforming link shall have precedence over an enabling link as shown in Figure 56.

Within the enabling links, an agent link shall have precedence over an instrument link because in artificial systems the humans are central to the process, they must ensure the system's proper operation. In addition, wherever there is human interaction, an interface should exist and this information should be available to the modeller of a refineable so that they can plan accordingly.

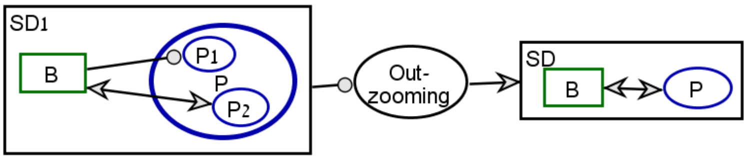


Figure 56 — Link precedence for transforming and enabling links

Summarizing the semantic strength of the procedural non-control links, the primary order of precedence shall be: consumption = result > effect > agent > instrument, where the = and > refer to the semantic strength of the links. State-specified links shall have higher precedence than basic links that do not specify states.

14.2.4.1.3 Secondary precedence among same-kind non-control links and control links

Each non-control link kind has a corresponding event and condition link that are useful for determining finer, secondary precedence distinction within each kind of procedural link. The relative semantic strength for the secondary order of precedence within each member of the primary order of precedence shall have the event link of stronger semantic strength than its corresponding non-control link, while the condition link shall have a weaker semantic strength than its corresponding non-control link.

The semantic strength of an event link shall be stronger than the semantic strength of its corresponding non-control link because any event link has semantics of both its corresponding non-control link plus the event capable of initiating a process. The semantic strength of a conditional link shall be weaker than the semantic strength of its corresponding non-control link because the condition modifier weakens the precondition satisfaction criteria for the connecting process.

14.2.4.1.4 Summary of the procedural links semantic strength

Summarizing the semantic strength of the procedural links based on the distinction between primary and secondary precedence, the complete order of precedence shall be:

1.	consumption event	>	consumption
2.	consumption	=	result
3.	result	>	consumption condition
4.	consumption condition	>	effect event
5.	effect event	>	effect
6.	effect	>	effect condition
7.	effect condition	>	agent event
8.	agent event	>	agent
9.	agent	>	agent condition
10.	agent condition	>	instrument event
11.	instrument event	>	instrument
12.	instrument	>	instrument condition

Annex A
(normative)

OPL Formal syntax in EBNF

A.1 Introduction

Object-Process Language (OPL) is a subset of English that shall express textually the OPM specification that the OPD set expresses graphically.

OPL is a dual-purpose language. First, it serves domain experts and system architects engaged in analyzing and designing a system, such as an electronic commerce system or a Web-based enterprise resource planning system. Second, it provides a firm basis for automatically generating the designed application.

OPL is the textual counterpart of the graphic OPM system specification, corresponding to the diagrammatic description in the OPD set. OPL shall be an automatically generated textual description of the system in a subset of natural English. Devoid of the idiosyncrasies and excessive cryptic details that characterize programming languages, OPL sentences shall be understandable to people without technical or programming experience.

Because of the extensive variety in model expression enabled by OPM, the OPL syntax expression in EBNF below is necessarily incomplete, e.g. the opportunities for statements regarding probability in 12.7 and execution path management in 13 are lacking EBNF expressions. The enormous variety of participation constraints, especially those expressible as mathematical formulas, do not have formal specification in Annex A.

A.2 OPL in the context of OPD

This Annex provides a formal specification of the Object-Process Language conforming to ISO 19477:1996, which results from the various OPD graphical constructions found in Clause 7 through Clause 14. To aid the reader, this Annex references the corresponding OPD sub-clauses where appropriate and Annex headings help to partition the EBNF according to syntactic forms for modelling elements..

NOTE With appropriate use of the graph grammar described in Annex C, and the symbols described in Annex A, sentences constructed in OPL are translatable into OPD figures.

A.3 Preliminaries

A.3.1 EBNF syntax

The following syntax uses the notation of EBNF as described in ISO 14977:1996[1]. The normal character representing each operator of Extended BNF and its implied precedence shall be (highest precedence at the top):

* repetition-symbol
- except-symbol
, concatenate-symbol
| definition-separator-symbol
= defining-symbol
; terminator-symbol

[1] ISO 14977 is a freely available standard that can be downloaded free of charge from http://isotc.iso.org/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards.htm

The normal precedence shall be over-ridden by the following bracket pairs:

```
‘ first-quote-symbol ’
“ second-quote-symbol  ”
(* start-comment-symbol end-comment-symbol *)
( start-group-symbol end-group-symbol )
[ start-option-symbol end-option-symbol ]
{ start-repeat-symbol end-repeat-symbol }
? special-sequence-symbol ?
```

NOTE 1 A space character enclosed in quotes as in “ “ denotes that a literal space character is required, otherwise space characters and line endings (so-called white space) have no significance.

NOTE 2 A meta identifier can occur on both the left and right sides of a rule, so enabling recursion.

NOTE 3 The first-quote-symbol identifies syntactic elements of OPL variable labels, which are the names and values appearing in OPD graphical models and OPL sentences. These particular syntactic elements are found only in the Base declarations subclause below.

NOTE 4 The second-quote-symbol identifies syntactic elements of OPL constants, which are words and phrases appearing in OPL sentences as interpretations of the graphical element configurations and link tags in an OPD.

NOTE 5 Beginning with A.3.2 and through the remainder of Annex A, all text, except headings, conform to ISO 14977.

A.3.2 Base declarations

```
(* Region OPL EBNF *)
(* Region Base declarations: The following base declarations define certain strings: *)

non zero digit = ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’ ;
decimal digit = ‘0’ | non zero digit ;
positive integer = non zero digit, {decimal digit} ;
positive real number = {decimal digit}, ".", decimal digit, {decimal digit} ;
upper case letter = ‘A’ | ‘B’ | ‘C’ | ‘D’ | ‘E’ | ‘F’ | ‘G’ | ‘H’ | ‘I’ | ‘J’ | ‘K’ | ‘L’ | ‘M’
| ‘N’ | ‘O’ | ‘P’ | ‘Q’ | ‘R’ | ‘S’ | ‘T’ | ‘U’ | ‘V’ | ‘W’ | ‘X’ | ‘Y’ | ‘Z’ ;
lower case letter = ‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘g’ | ‘h’ | ‘i’ | ‘j’ | ‘k’ | ‘l’ | ‘m’
| ‘n’ | ‘o’ | ‘p’ | ‘q’ | ‘r’ | ‘s’ | ‘t’ | ‘u’ | ‘v’ | ‘w’ | ‘x’ | ‘y’ | ‘z’ ;
letter = upper case letter | lower case letter ;
string character = letter | decimal digit | ‘_’ | ‘-‘ | '&' | ‘/’ | ‘ ‘ ;  (* note that a string character can be a space *)
name = letter, {string character} ;                  (* note that the first character is a letter *)
capitalized word = upper case letter {string character} ;
non capitalized word = lower case letter {string character} ;
non capitalized phrase = non capitalized word, { ' ', ( non capitalized word | capitalized word ) } ;
type identifier = " boolean"
                  | " string"
                  | number type
                  | " enumerated" ;
prefix = " unsigned" ;
number type = [prefix], " integer"
                  | " float"
                  | " double"
                  | " short"
                  | " long" ;
participation limit = positive integer | positive real number ;
participation constraint = lower single
                  | upper single
                  | lower plural
                  | upper plural
                  | ( "0" | participation limit, [ " to ", participation limit ]  ) ;
```



```
expression constraint = " where ", name, ( ( logical operation, value name )
                    | ( logical begin set, ( name | value name ), { ", ", [ ( name | value name ) ] },
                        logical end set ) ) ;
lower single = "a " | "an " | "an optional " | "at least one " ;
upper single = "A " | "An " | "An optional " | "At least one " ;
lower plural = "optional " | "many " ;
upper plural = "Optional " | "Many " ;
range clause = " is ", value name | " ranges from ", value name, " to ", value name ;
logical operation = "=" | "<" | ">" | "<=" | " >=" ;
logical begin set = " in { " ;
logical end set = " }" ;
```

(* participation constraints have many forms of expression and the Base Declarations do not include all of those forms. *)

(* Reserved words and symbols found in OPL statements are delimited by second quote symbols *)

(* EndRegion: Base declarations *)

A.3.3 OPL special sequences

(* Region: special sequences – This region defines all special sequences like New Line, Plural objects and processes *)

```
new line = ? application specific character sequence resulting in a line feed followed by return to first character position on the line ? ;
measurement unit = ? any specified or commonly understood measurement of time, space, quantity, or quality? ;
value name = ? a number or name appropriate for the associated measurement unit? ;
singular object name = ? capitalized singular noun phrase ? ;                                    (* see 7.1.2 *)
plural object name = ? capitalized plural noun phrase ? ;
singular process name = ? capitalized gerund phrase ? | ? capitalized singular noun phrase ? ;
plural process name = ? capitalized gerund phrase ? | ? capitalized plural noun phrase ? ;        (* see 7.2.2 *)
parent OPD = ? OPD from which a new-diagram in-zooming or new diagram unfolding occurs ? ;
child OPD = ? OPD resulting from a new-diagram in-zooming or new diagram unfolding ? ;
max duration time units = ? value of maximum duration in time units for process execution ? ;
min duration time units = ? value of minimum duration in time units for process execution ? ;
```

(* EndRegion: Special Sequences *)

A.4 OPL Syntax

A.4.1 OPL document structure

(* Region OPL document *)

```
OPL paragraph = OPL sentence, { new line, OPL sentence} ;
OPL sentence = OPL formal sentence, "." ;
OPL formal sentence = thing description sentence
                    | procedural sentence
                    | structural sentence
                    | context management sentence ;
```

A.4.2 OPL Identifiers

(* Region: Identifiers – This region defines all identifiers used throughout the grammar *)

```
object identifier = singular object name, [" in ", measurement unit], [range clause]
```

```
                    | singular object name, " object", [" in ", measurement unit], [range clause]
                    | plural object name, [" in ", measurement unit], [range clause]
                    | plural object name, " objects", [" in ", measurement unit], [range clause]  ;
process identifier = singular process name
                    | singular process name, " process"
                    | plural process name
                    | plural process name, " processes" ;
thing identifier = object identifier
                    | process identifier ;                                    (* see 7.1 and 7.2 *)
state identifier = non capitalized word ;
tag expression = non capitalized phrase ;

(* EndRegion: Identifiers *)
```

A.4.3 OPL lists

```
(* Region: Lists – This region defines various lists: object list, process list, object with optional state list *)

process list = process identifier
                    | process identifier, [ {", ", process identifier} ], " and ", process identifier ; (* see 12.1 *)
process Or list = process identifier, [{", ", process identifier} ], " or ", process identifier ;
process Xor list at beginning = "One of ", process Or list ;
process Xor list at end = "one of ", process Or list ;

object list = object identifier
                    | object identifier, [ {", ", object identifier} ], " and ", object identifier ;      (* see 12.1 *)
object with optional state = [state identifier], " ", object identifier ;
(* object with optional state may replace object identifier in many OPL expressions using object identifier *)
object with optional state list = object with optional state
                    | object with optional state, [ {", ", object with optional state} ],
                              " and ", object with optional state ;

object Or list = object with optional state, [ {", ", object with optional state} ], " or ", object with optional state ;
                                                                                          (* see 12.2 *)
object Or list nostates = object identifier, [ {", ", object identifier} ], " or ", object identifier ;

object Xor list at beginning = "One of ", object Or list ;
object Xor list at end = "one of ", object Or list ;
object nostates Xor list at end = "one of ", object Or list ;

state list = state identifier
                    | state identifier, [ {", ", state identifier} ], " and ", state identifier ;
state Or list = state identifier, [ {", ", state identifier} ], " or ", state identifier ;
state Xor list at end = "one of ", state Or list ;

(* EndRegion: Lists *)
```

A.4.4 OPL Thing description

A.4.4.1 Thing description sentence

```
(* Region: Thing Description – This region defines all thing description sentences *)

thing description sentence = generic property sentence
                    | type description sentence
                    | state description sentence ;
```

A.4.4.2 Generic property sentence

```
generic property sentence = thing identifier,
        " is ", [ essence ], [ affiliation ], [ persistence ] ;        (* see 7.3.3 *)
essence = "Informatical" | "Physical" ;        (* Physical is the non-default value of
                                                Essence, the default value of which is
                                                Informatical. *)
affiliation = "Systemic" | "Environmental" ;  (* Environmental is the non-default
                                                value of Affiliation, the default value
                                                of which is Systemic. *)
persistence = "Persistent" | "Transient" ;    (* Transient is the non-default value
                                                of Persistence, the default value of
                                                which is Persistent. *)
```

A.4.4.3 Type description sentence

```
type description sentence = object identifier, " is of type ", type identifier ;
```

A.4.4.4 State description sentence

```
state description sentence = state enum sentence
        | initial states sentence
        | final states sentence
        | default state sentence
        | combined state sentence ;        (* see 7.3.5 *)
state enum sentence = object identifier, " is ", state identifier
        | object identifier, " can be ",
            state identifier, [{", ", state identifier}], " and ", state identifier
        | object identifier, " can be ",
            state identifier, [{", ", state identifier}], " and other states"  ;
initial states sentence = single initial states sentence
        | multiple initial states sentence ;
single initial states sentence = "State ", state identifier, " of ", object identifier, " is initial" ;
multiple initial states sentence = "States ", state list " of ", object identifier, " are initial" ;
final states sentence = single final state sentence
        | multiple final state sentence ;
single final state sentence = "State ", state identifier, " of ", object identifier, " is final" ;
multiple final state sentence = "States ", state list, " of ", object identifier, " are final" ;
default state sentence = "State " state identifier, " of ", object identifier, " is default" ;
combined state sentence = object identifier, {" is initially ", [state identifier | state identifier,
            {" and ", state identifier}], " and finally ", state OR list } ;
input state = state identifier ;    (* the state or states of the associated object in a process precondition set *)
output state = state identifier ;   (* the state or states of the associated object in a process postcondition set *)

active process identifier = process identifier ;

(* EndRegion: Thing Description *)
```

A.4.5 OPL Procedural sentences

A.4.5.1 Procedural sentnece

```
(* Region: Procedural sentences. – This region defines all procedural sentences *)

procedural sentence = transforming sentence
        | enabling sentence
        | control sentence ;        (* see 8.1.1 *)
```

A.4.5.2 OPL Transformations

A.4.5.2.1 Transforming sentence

(* Region: Transforming sentences – This region defines consumption, result, effect and change sentences, and their variations *)

```
transforming sentence = consumption sentence
                        | result sentence
                        | effect sentence
                        | change sentence ;                                        (* see 9.1.1 and 9.3.3 *)
```

A.4.5.2.2 Consumption sentence

```
consumption sentence = ( process identifier, " consumes ", object with optional state list )
                        | consumption select sentence ;                                    (* see 9.1.2 *)
consumption select sentence = consumption Or sentence
                        | consumption Xor sentence ;                                       (* see 12.3 *)
consumption Or sentence = consumption source Or sentence
                        | consumption destination Or sentence ;
consumption source Or sentence = process identifier, " consumes at least one of ", object Or list ;
consumption destination Or sentence = "At least one of ", process Or list,
                            " consumes ", object with optional state ;

consumption Xor sentence = consumption source Xor sentence
                        | consumption destination Xor sentence ;
consumption source Xor sentence = process identifier, " consumes exactly ", object Xor list at end ;
consumption destination Xor sentence = "Exactly ", process Xor list at beginning, " consumes ",
                            object with optional state ;
```

A.4.5.2.3 Result sentence

```
result sentence = (process identifier, " yields ", object with optional state list )
                        | result select sentence ;                                         (* see 9.1.3 *)
result select sentence = result Or sentence
                        | result Xor sentence ;                                            (* see 12.3 *)
result Or sentence = result source Or sentence
                        | result destination Or sentence ;
result source Or sentence = "At least one of ", process Or list, " yields ", object with optional state ;
result destination Or sentence = process identifier, " yields at least one of ", object Or list ;
result Xor sentence = result source Xor sentence
                        | result destination Xor sentence ;
result source Xor sentence = "Exactly ", process Xor list at beginning, " yields ", object with optional state ;
result destination Xor sentence = process identifier, " yields exactly ", object Xor list at end ;
```

A.4.5.2.4 Effect sentence

```
effect sentence = (process identifier, " affects ", object list )
                        | effect select sentence ;                                         (* see 9.1.4 *)
effect select sentence = effect Or sentence
                        | effect Xor sentence ;
effect Or sentence = effect object Or sentence
                        | effect process Or sentence ;                                     (* see 12.3 *)
effect object Or sentence = process identifier, " affects  at least one of ", object Or list Nostates ;
effect process Or sentence = "At least one of ", process Or list, " affects ", object identifier ;
effect Xor sentence = effect object Xor sentence
                        | effect process Xor sentence ;
effect object Xor sentence = process identifier, " affects exactly ", object nostates Xor list at end ;
effect process Xor sentence = "Exactly ", process Xor list at beginning, " affects ", object identifier ;
```

A.4.5.2.5 Change sentence

```
change sentence = in out specified change sentence
                | input specified change sentence
                | output specified change sentence ;                          (* see 9.3.3.1 *)

in out specified change sentence = ( process identifier, " changes ", in out object change list )
                | in out specified change select sentence ;                   (* see 9.3.3.2 *)
in out object change list = in out object change phrase
                | in out object change phrase, [ {", ", in out object change phrase} ],
                   " and ", in out object change phrase ;
in out object change phrase = object identifier, " from ", input state, " to ", output state ;
in out specified change select sentence = in out specified change Or sentence
                | in out specified change Xor sentence ;
in out specified change Or sentence = (process identifier, " changes ", Or in out object change list )
                | ( process Or list, " changes ", in out object change phrase )
                | in out specified change state Or sentence ;
Or in out object change list = in out object change phrase, [ {", ", in out object change phrase} ],
                   " or ", in out object change phrase ;
in out specified change state Or sentence = ( process identifier, " changes ", object identifier,
                   " from ", state Or list, " to ", state identifier )
                | ( process identifier, " changes ", object identifier,
                   " from ", state identifier, " to ", state Or list ) ;
in out specified change Xor sentence = in out specified change object Xor sentence
                | in out specified change process Xor sentence
                | in out specified change state Xor sentence ;
in out specified change Object Xor sentence = process identifier, " changes one of ",
                          Or In out object change list ;

in out specified change process Xor sentence = process Xor list at beginning, " changes ",
                   in out object change phrase ;
in out specified change state Xor sentence = ( process identifier, " changes ", object identifier,
                   " from ", state Xor list at end, " to ", state identifier )
                | (process identifier, " changes ", object identifier, " from ", state identifier, " to ",
                   state Xor list at end ) ;

input specified change sentence = ( process identifier, " changes ", input object change list )
                | input specified change select sentence ;                    (* see 9.3.3.3 *)
input object change phrase = object identifier, " from ", input state ;
input object change list = input object change phrase
                | input object change phrase, [ {", ", input object change phrase } ], " and ",
                   input object change phrase ;
input specified change select sentence = input specified change Or sentence
                | input specified change Xor sentence ;
input specified change Or sentence = ( process identifier, " changes ", Or input object change list )
                | (process Or list, " changes ", input object change phrase )
                | (process identifier, " changes ", object identifier, " from ", state Or list ) ;
Or input object change list = input object change phrase, [ {", ", input object change phrase } ], " or ",
                   input object change phrase ;
input specified change Xor sentence = (process identifier, " changes one of ", Or input object change list )
                | (process Xor list at beginning, " changes ", input object change phrase )
                | (process identifier, " changes ", object identifier, " from ", state Xor list at end ) ;

output specified change sentence = (process identifier, " changes ", output object change list )
                | output specified change select sentence ;                   (* see 9.3.3.4 *)
output object change list = output object change phrase
                | output object change phrase, [ {", " output object change phrase } ], " and ",
                   output object change phrase ;
output object change phrase = object identifier, " to ", output state ;
output specified change select sentence = output specified change Or sentence
```

```
                    | output specified change Xor sentence ;
output specified change Or sentence = (process identifier, " changes ", Or output object change list )
                    | (process Or list, " changes ", output object change list )
                    | (process identifier, " changes ", object identifier, " to ", state Or list ) ;
Or output object change list = output object change phrase, [ {", ", output object change phrase } ], " or ",
                        output object change phrase ;
output specified change Xor sentence = (process identifier, " changes one of ", Or output object change list )
                    | (process Xor list at beginning, " changes ", output object change phrase )
                    | process identifier, " changes ", object identifier, " to ", state Xor list at end ;

(* EndRegion: Transforming sentences *)
```

A.4.5.3 OPL Enablers

A.4.5.3.1 Enabling sentences

(* Region: Enabling sentences – This region defines Agent and Instrument sentences and their possible variations *)

```
enabling sentence = agent sentence
                    | instrument sentence ;                                   (* see 9.2.1 *)
```

A.4.5.3.2 Agent sentence

```
agent sentence = (object with optional state list, " handle ", process identifier )
                    | agent select sentence ;                           (* see 9.2.2 and 12.3 *)

agent select sentence = agent Or sentence
                    | agent Xor sentence ;
agent Or sentence = agent source Or sentence
                    | agent destination Or sentence ;
agent source Or sentence = "At least one of ", object Or list, "handles", process identifier ;
agent destination Or sentence = object with optional state, "handles at least one of ", process Or list ;
agent Xor sentence = agent source Xor sentence
                    | agent destination Xor sentence ;
agent source Xor sentence = "Exactly ", object Xor list at beginning, " handles ", process identifier ;
agent destination Xor sentence = object with optional state, " handles exactly ", process Xor list at end ;
```

A.4.5.3.3 Instrument sentence

```
instrument sentence = (process identifier, " requires ", object with optional state list )
                    | instrument select sentence ;                      (* see 9.2.3 and 12.3 *)

instrument select sentence = instrument Or sentence
                    | instrument Xor sentence ;
instrument Or sentence = instrument source Or sentence
                    | instrument destination Or sentence ;
instrument source Or sentence = process identifier, " requires at least one of ", object Or list ;
instrument destination Or sentence = "At least one of ", process Or list, " requires ", object with optional state ;
instrument Xor sentence = instrument source Xor sentence
                    | instrument destination Xor sentence ;
instrument source Xor sentence = process identifier, " requires exactly ", object Xor list at end ;
instrument destination Xor sentence = "Exactly ", process Xor list at beginning, " requires ", object with optional state ;

(* EndRegion: Enabling sentences *)
```

A.4.5.4 OPL Flow of control

A.4.5.4.1 Control sentence

```
(* Region : Control sentences – This region defines all sentences related to flow of control in the system *)

control sentence = event sentence
                 | condition sentence
                 | invocation sentence
                 | exception sentence ;                                   (* see 9.5.1 *)
```

A.4.5.4.2 Event sentence

```
event sentence = consumption event sentence
               | effect event sentence
               | agent event sentence
               | instrument event sentence ;                              (* see 9.5.2 *)

consumption event sentence = object with optional state, " initiates ", process identifier,
                   ", which consumes ", object identifier ;
                                   (* see 12.5 and 12.6 for additional syntax for link fans *)
effect event sentence = simple effect event sentence
               | in out specified effect event sentence
               | input specified effect event sentence
               | output specified effect event sentence ;

simple effect event sentence = object identifier, " initiates ", process identifier, ", which affects ",
                   object identifier ;
in out specified effect event sentence = input state, object identifier, " initiates ", process identifier,
                   ", which changes ", in out object change phrase ;
input specified effect event sentence = input state, object identifier, " initiates ", process identifier,
                   ", which changes ", object identifier, " from ", input state ;
output specified effect event sentence =  object identifier, " in any state initiates ", process identifier,
                   ", which changes ", object identifier, " to ", output state ;

agent event sentence = object with optional state, " initiates and handles ", process identifier ;
instrument event sentence = object with optional state, " initiates ", process identifier,
                    ", which requires " object with optional state ;
```

A.4.5.4.3 Condition sentence

```
condition sentence = condition transforming sentence
               | condition enabling sentence ;
condition transforming sentence =  conditional consumption sentence
               | conditional state specified consumption sentence
               | conditional effect sentence
               | conditional state specified consumption sentence ;      (* see 9.5.3.1 and 9.5.3.3 *)

conditional consumption sentence = ( process identifier, " occurs if ", object identifier,
                   " exists, in which case ", object identifier, " is consumed, otherwise
                   ", process identifier, " is skipped " )
               | ( "If ", object identifier, " exists then ", process identifier, " occurs and consumes ",
                   object identifier, ", otherwise bypass ", process identifier ) ;
conditional state specified consumption sentence = ( process identifier, " occurs if ", object identifier,
                   " is ", input state, ", in which case ", object identifier, " is consumed, otherwise
                   ", process identifier, " is skipped " )
               | ( "If ", input state, object identifier, " exists then ", process identifier,
                   " occurs and consumes ", object identifier, ", otherwise bypass ",
                    process identifier ) ;
```

```
conditional effect sentence = simple conditional effect sentence
                    | in out specified conditional effect sentence
                    | input specified conditional effect sentence ;
simple conditional effect sentence = ( process identifier, "occurs if ", object identifier,
                        " exists, in which case ", process identifier, " affects ", object identifier,
                        ", otherwise ", process identifier, " is skipped " )
                    | ( "If ", object identifier, " exists then ", process identifier, "occurs and affects ",
                        object identifier, ", otherwise bypass ", process identifier ) ;
in out specified conditional effect sentence = ( process identifier, " occurs if there is ",
                        input state, object identifier, ", in which case ", process identifier, " changes ",
                        in out object change phrase, ", else ", process identifier,
                        " is skipped " )
                    | ( process identifier, " occurs if there is ",
                        input state, object identifier, ", in which case ", process identifier, " changes ",
                        in out object change phrase,
                        ", otherwise bypass ", process identifier ) ;
input specified conditional effect sentence = (process identifier, " occurs if there is ",
                        input state, object identifier, " in which case ", process identifier, " changes ",
                        object identifier, " from ", Input state, ", else ", process identifier, " is skipped " )
                    | (process identifier, " occurs if there is ", input state, object identifier,
                        " in which case ", process identifier, " changes ", object identifier, " from ",
                        Input state, ", otherwise bypass ", process identifier ) ;

condition enabling sentence = conditional agent sentence
                    | conditional instrument sentence ;                    (* see 9.5.3.2 *)
conditional agent sentence = ( process identifier, " occurs if ", object with optional state,
                        " exists, else ", process identifier, " is skipped" )
                    | ( process identifier, " occurs if ", object with optional state,
                        " exists, else bypass ", process identifier ) ;
conditional instrument sentence = ( process identifier, " occurs if ", object with optional state,
                        " exists, else ", process identifier, " is skipped" )
                    | ( process identifier, " occurs if ", object with optional state,
                        " exists, else bypass ", process identifier ) ;
```

A.4.5.4.4 Invocation sentence

```
invocation sentence = (process identifier, " invokes ", process list )
                    | (process identifier, " invokes itself " )
                    | invocation select sentence ;              (* see 9.5.2.5 and 12.3 *)

invocation select sentence = invocation Or sentence
                    | invocation Xor sentence ;

invocation Or sentence = ( "At least one of ", process Or list, " invokes ", process identifier )
                    | (process identifier, " invokes at least one of", process Or list ) ;
invocation Xor sentence = ( "Exactly one of ", process Or list, " invokes ", process identifier )
                    | (process identifier, " invokes exactly ", process Xor list at end );
```

A.4.5.4.5 Exception sentence

```
exception sentence = overtime exception sentence
                    | undertime exception sentence ;                    (* see 9.5.4 *)
overtime exception sentence = active process identifier, " occurs if duration of ", process identifier, " exceeds ",
                        max duration time units ;
undertime exception sentence = active process identifier, " occurs if duration of ", process identifier,
                        " falls short of ", min duration time units ;

(* EndRegion: Control sentences *)
(* EndRegion: Procedural sentences *)
```

A.4.6 OPL Structural sentences

A.4.6.1 Structural sentence

(* Region: Structural sentences - This region defines all sentences that connect things in static, time-independent, long-lasting relations *)

```
structural sentence = tagged structural sentence
                    | aggregation sentence
                    | characterization sentence
                    | exhibition sentence
                    | specialization sentence
                    | instantiation sentence ;                      (* see 10.1 *)
```

A.4.6.2 OPL tagged structures

A.4.6.2.1 Tagged structural sentence

```
tagged structural sentence = unidirectional tagged structural sentence
                    | bidirectional tagged structural sentence ;
```

A.4.6.2.2 Unidirectional tagged structural sentence

```
unidirectional tagged structural sentence = single link unidirectional tagged sentence
                    | forked tagged structural sentence ;           (* see 10.2.1 and 11.2 *)
single link unidirectional tagged sentence = nullTag unidirectional object tagged structural sentence
                    | nullTag unidirectional process tagged structural sentence
                    | non nullTag unidirectional object tagged structural sentence
                    | non nullTag unidirectional process tagged structural sentence ;
                                                                    (* see 10.2.2 and 11.2 *)

nullTag unidirectional object tagged structural sentence = [participation constraint, " "],
                        source object, uniDirNullTag, [participation constraint, " "], destination object ;
nullTag unidirectional process tagged structural sentence = [participation constraint, " "],
                        source process, uniDirNullTag, [participation constraint, " "], destination process ;
non nullTag unidirectional object tagged structural sentence = [participation constraint, " "], source object, " ",
                        forward tag, " ", [participation constraint, " "], destination object,
                        [expression constraint] ;
non nullTag unidirectional process tagged structural sentence = [participation constraint, " "], source process,
                        " ", forward tag, " ", [participation constraint, " "], destination process ;

forked tagged structural sentence = forked nullTag object tagged structural sentence
                    | forked nullTag process tagged structural sentence
                    | forked non nullTag object tagged structural sentence
                    | forked non nullTag process tagged structural sentence ;
forked nullTag object tagged structural sentence = [participation constraint, " "], source object, uniDirNullTag,
                        object tine set ;
forked nullTag process tagged structural sentence = [participation constraint, " "], source process,
                        uniDirNullTag, process tine set ;
forked non nullTag object tagged structural sentence = [participation constraint, " "], source object, " ",
                        forward tag, " ", object tine set ;
forked non nullTag process tagged structural sentence = [participation constraint, " "], source process, " ",
                        forward tag, " ", process tine set ;

object tine set = tine object | ( ( tine object, [ {", ", tine object } ], " and ", ( tine object | "more" ) ),
                        [ ( ", ordered by ", order criteria ) | ( ", in that sequence" ) ] ) ;
process tine set =  tine process | ( ( tine process, [ {", ", tine process } ], " and ", ( tine process | "more" ) ),
                        [ ( ", ordered by ", order criteria ) | ( ", in that sequence" ) ] ) ;
order criteria = name ;
```

```
tine object = [ participation constraint, " " ], object with optional state ;
source object = object with optional state ;
destination object = object with optional state ;
tine process = [ participation constraint, " " ], process identifier ;
source process = process identifier ;
destination process = process identifier ;
uniDirNullTag = " relates to "
                    | " relate to "
                    | user defined uniDirNullTag ;
forward tag = tag expression ;
user defined uniDirNullTag = tag expression ;
```

A.4.6.2.3 Bidirectional tagged structural sentences

```
bidirectional tagged structural sentence = asymmetric bidirectional object tagged structural sentence
                    | asymmetric bidirectional process tagged structural sentence
                    | symmetric bidirectional object tagged structural sentence
                    | symmetric bidirectional process tagged structural sentence ; (* see 10.2.3 and 11.2 *)

asymmetric bidirectional object tagged structural sentence =
                        ( [ participation constraint, " " ], source object, bidir forward tag,
                            [ participation constraint, " " ], destination object, [expression constraint] )
                    | ( [ participation constraint, " " ], destination object, bidir backward tag,
                        [ participation constraint, " " ], source object, [expression constraint] ) ;
asymmetric bidirectional process tagged structural sentence =
                        ( [ participation constraint, " " ], source process, bidir forward tag,
                            [ participation constraint, " " ], destination process )
                    | ( [ participation constraint, " " ], destination process, bidir backward tag,
                        [ participation constraint, " " ], source process ) ;
symmetric bidirectional object tagged structural sentence =
                        ( [ participation constraint, " " ], source object, " and ", [ participation constraint, " " ],
                        destination object, " are ", biDirNullTag )
                    | ( [ participation constraint, " " ], source object, " and ",
                            [ participation constraint, " " ],
                             destination object ), " are ", symmetric tag ;
symmetric bidirectional process tagged structural sentence =
                        ( [ participation constraint, " " ], source process,
                        " and ", [ participation constraint, " " ], destination process, " are ", biDirNullTag )
                    | ( [ participation constraint, " " ], source process,
                        " and ", [ participation constraint, " " ], destination process ), " are ", symmetric tag ;

symmetric tag = tag expression ;
bidir forward tag = tag expression ;
bidir backward tag = tag expression ;
biDirNullTag = " related"
                    | user defined biDirNullTag ;
user defined biDirNullTag = tag expression ;
```

A.4.6.3 OPL fundamental structures

A.4.6.3.1 Aggregation sentences

```
aggregation sentence = object forked aggregation sentence
                    | process forked aggregation sentence ;                    (* see 10.3.2 *)
object forked aggregation sentence = whole object, " consists of ", object parts list ;
process forked aggregation sentence = whole process, " consists of ", process parts list ;
object parts list = part object
                    | (part object, [ { ", ", part object } , " and ", ( part object | " at least one other part" ) ] ) ;
process parts list = part process
                    | (part process, [ { ", ", part process }, " and ",
```

```
                                  ( part process | " at least one other part" ) ] ) ;
whole object = object identifier ;
part object = [participation constraint, " "], object identifier ;
whole process = process identifier ;
part process = [participation constraint, " "], process identifier ;
```

A.4.6.3.2 Characterization sentences

```
characterization sentence = object forked characterization sentence
                            | process forked characterization sentence ;          (* see 10.3.3 *)

object forked characterization sentence = basic object forked characterization sentence
                            | partial object forked characterization sentence
                            | AsWellAs object forked characterization sentence
                            | partial AsWellAs object forked characterization sentence ;
basic object forked characterization sentence = object identifier, " exhibits ", ( attribute list | operator list ) ;
partial object forked characterization sentence = object identifier, " exhibits ", ( (attribute list,
                                  ", and at least one other attribute " ) | ( operator list,
                                  ", and at least one other operator" )) ;
AsWellAs object forked characterization sentence = object identifier, " exhibits ", attribute list, ", as well as ",
                                  operator list ;
partial AsWellAs object forked characterization sentence = object identifier, " exhibits ", attribute list,
                                  ", and at least one other attribute", ", as well as ", operator list,
                                  ", and at least one other operator" ;

attribute = object identifier ;
operator = process identifier ;
attribute list = object list ;
operator list = process list ;

process forked characterization sentence = basic process forked characterization sentence
                            | partial process forked characterization sentence
                            | partial AsWellAs process forked characterization sentence
                            | AsWellAs process forked characterization sentence ;
basic process forked characterization sentence = process identifier, " exhibits ", ( operator list | attribute list ) ;
partial process forked characterization sentence = process identifier, " exhibits ", ( (operator list,
                                  ", and at least one other operator " ) | ( attribute list,
                                  ", and at least one other attribute" )) ;

AsWellAs process forked characterization sentence = process identifier, " exhibits ", operator list, ",
                                  as well as ", attribute list ;
partial AsWellAs process forked characterization sentence = process identifier, " exhibits ", operator list,
                                  ", and at least one other operator", ", as well as ", attribute list,
                                  ", and at least one other attribute ;
```

A.4.6.4 Exhibition sentences

```
exhibition sentence = object exhibition sentence
                            | process exhibition sentence ;          (* see 10.3.3.2.2 and 11.3 *)
object exhibition sentence = feature, " of ", object identifier, ( range clause | " is ",
                                  ( ( attribute list | operator list ) | ( attribute list, " as well as ", operator list ) ) ) ;
process exhibition sentence = feature, " of " , process identifier, " is ", ( ( operator list | object list )
                                  | ( operator list, " as well as ", attribute list ) ) ;

feature = attribute | operator ;
```

A.4.6.5 Specialization sentences

```
specialization sentence = object specialization sentence
                            | process specialization sentence
```

```
                        | state specialization sentence ;                                   (* see 10.3.4 *)

object specialization sentence = basic object specialization sentence
                        | multiple object specialization sentence
                        | partial object specialization sentence
                        | Xor object specialization sentence
                        | multiple object inheritance specialization sentence ;

basic object specialization sentence = special object, " is a ", general object ;
multiple object specialization sentence = special object list, " are ", general object ;
partial object specialization sentence = special object list, " and other specializations are ", general object ;
Xor object specialization sentence = basic Xor object specialization sentence
                        | comma separated Xor object specialization sentence ;
basic Xor object specialization sentence = special object, " can be either ", general object, " or ",
                            general object ;
comma separated Xor object specialization sentence = special object, " can be one of ", general object,
                            { ", ", general object }, " or ", general object ;
multiple object inheritance specialization sentence = special object, " is ", general object list ;

general object = object identifier ;
special object = object identifier ;
general object list  =  " a ", object identifier, [ { " a ", object identifier } ], " and a ", object identifier ;
special object list = object list ;

process specialization sentence =basic process specialization sentence
                        | multiple process specialization sentence
                        | partial process specialization sentence
                        | Xor process specialization sentence
                        | multiple process inheritance specialization sentence ;
basic process specialization sentence = special process, " is ", general process ;
multiple process specialization sentence = special process list, " are ", general process ;
partial process specialization sentence = special process list, " and other specializations are ",
                            general process ;
Xor process specialization sentence = basic Xor process specialization sentence
                        | comma separated Xor process specialization sentence ;
basic Xor process specialization sentence = special process, " can be either ", general process, " or ",
                            general process ;
comma separated Xor process specialization sentence = special process, " can be one of ", general process,
                            { ", ", general process }, " or ", general process ;
multiple process inheritance specialization sentence = special process, " is ", general process list ;

general process = process identifier ;
special process = process identifier ;
general process list = " a", process identifier, [ { " a ", process identifier } ] " and a ", process identifier  ;
special process list = process list ;

state specialization sentence = basic state specialization sentence
                        | multiple state specialization sentence
                        | partial state specialization sentence ;
basic state specialization sentence = state specified object, " is a ", state specified object ;
multiple state specialization sentence = state specified object list, " are ", state specified object ;
partial state specialization sentence = state specified object list, " and other specializations are
                            ", state specified object ;

state specified object = state identifier, " ", object identifier ;
state specified object list = state specified object
                        | state specified object, [ { ", ", state specified object } ], " and ", state specified object ;
```

A.4.6.6 Instantiation sentences

```
instantiation sentence = object instantiation sentence
                        | process instantiation sentence ;                    (* see 10.3.5 *)

object instantiation sentence = basic object instantiation sentence
                        | multiple object instantiation sentence ;
basic object instantiation sentence= instance object, " is an instance of ", object class ;
multiple object instantiation sentence = instance object list, " are instances of ", object class ;

process instantiation sentence = basic process instantiation sentence
                        | multiple process instantiation sentence ;
basic process instantiation sentence = instance process, " is an instance of ", process class ;
multiple process instantiation sentence = instance process list, " are an instance of ", process class ;

instance object = object identifier ;
instance process = process identifier ;
object class = object identifier ;
process class = process identifier ;
instance object list = object list ;
instance process list = process list ;

(* EndRegion: Structural sentences *)
```

A.4.7 OPL Context management

A.4.7.1 Context management sentence

(* Region: Context management sentences - This region defines all sentences that manage OPD context shifts *)

```
context management sentence = unfolding sentence
                        | folding sentence
                        | in Zooming sentence
                        | out Zooming sentence ;                              (* see 14.2.1 *)

(* in diagram object and process unfolding are equivalent to corresponding structural sentences *)
```

A.4.7.2 Unfolding sentences

```
unfolding sentence = object unfolding sentence
                        | process unfolding sentence ;
object unfolding sentence = underspecified object unfolding sentence
                        | whole object unfolding sentence
                        | general object unfolding sentence
                        | class object unfolding sentence
                        | exhibitor object unfolding sentence ;

underspecified object unfolding sentence = object identifier, " unfolds into ", attribute list,
                            [ " as well as ", operator list ] ;
whole object unfolding sentence =  whole object, " from ", parent OPD, " part-unfolds in ", child OPD,
                            " into ", object parts list ;
general object unfolding sentence = general object, " from ", parent OPD, " specialization-unfolds in ",
                            child OPD, " into ", special object list ;
class object unfolding sentence = object class, " from ", parent OPD, " instance-unfolds in ", child OPD,
                            " into ", instance object list ;
exhibitor object unfolding sentence = object identifier, " from ", parent OPD, " feature-unfolds in ", child OPD,
                            " into ", attribute list, [ " as well as ", operator list ] ;
```

```
process unfolding sentence = underspecified process unfolding sentence
                        | whole process unfolding sentence
                        | general process unfolding sentence
                        | class process unfolding sentence
                        | exhibitor process unfolding sentence ;
underspecified process unfolding sentence = process identifier, " unfolds into ", operator list,
                             [", as well as ", attribute list] ;
whole process unfolding sentence = whole process, " from ", parent OPD, " part-unfolds in ", child OPD,
                             " into ", process parts list ;
general process unfolding sentence = general process, " from ", parent OPD, " specialization-unfolds in ",
                             child OPD, " into ", special process list ;
class process unfolding sentence = process class, " from ", parent OPD, " instance-unfolds in ", child OPD,
                             " into ", instance process list ;
exhibitor process unfolding sentence = process identifier, " from ", parent OPD, " feature-unfolds in ",
                             child OPD, " into ", operator list, [ " as well as ", attribute list ] ;
```

A.4.7.3 Folding sentences

```
folding sentence = object folding sentence
                        | process folding sentence ;

(* a folding sentence is only relevant for an OPD object or process for which unfolding produces a child OPD
and is the OPL equivalent to the graphical bold contour designation *)

object folding sentence = object identifier, " is folding of ", child OPD ;
process folding sentence = process identifier, " is folding of ", child OPD;
```

A.4.7.4 In zoom sentence

```
in zooming sentence = process in zoom sentence
                        | object in zoom sentence ;
process in zoom sentence = in diagram process in zoom sentence
                        | new diagram process in zoom sentence ;

in diagram process in zoom sentence = ( process identifier, " zooms into ", process list, "in that sequence",
                             [ ", as well as ", object in zoom list ] )
                        | ( process identifier, "  zooms into parallel ", process list, [ ", as well as ",
                             object in zoom list ] )
                        | ( process identifier, " zooms into ", process list, " and parallel ", process list,
                             ", in that sequence", [ ", as well as ", object in zoom list ] ) ;
new diagram process in zoom sentence = ( process identifier, " from ", parent OPD, " zooms in ", child OPD,
                             " into ", process list, "in that sequence", [ ", as well as ", object in zoom  list ] )
                        | ( process identifier, " from ", parent OPD, " zooms in ", child OPD, " into parallel ",
                             process list, [ ", as well as ", object in zoom list ] )
                        | ( process identifier, " from ", parent OPD, " zooms in ", child OPD, " into ",
                             process list, " and parallel ", process list, ", in that sequence",
                             [ ", as well as ", object in zoom list ] ) ;

object in zoom sentence = in diagram object in zoom sentence
                        | new diagram object in zoom sentence ;

in diagram object in zoom sentence = ( object identifier, " zooms into ", object list, "in that sequence",
                             [ ", as well as ", process in zoom list ] ) ;
new diagram object in zoom sentence = ( object identifier, " from ", parent OPD, " zooms in ", child OPD,
                             " into ", object list, "in that sequence", [ ", as well as ", process in zoom list ] ) ;

object in zoom list = object identifier, [ { ", ", object identifier }, " and ", object identifier, ", in that sequence" ] ;
process in zoom list = process identifier, [ {", ", process identifier }, " and ", process identifier,
                             ", in that sequence" ] ;
```

A.4.7.5 Out zooming sentence

```
out zooming sentence = process out zoom sentence
                     | object out zoom sentence ;
```

(* an out zoom sentence is only relevant for an OPD process or object for which in zooming produces a child OPD and is the OPL equivalent to the graphical bold contour designation *)

```
process out Zoom sentence = process identifier,  " is out zoom from ", child OPD ;
object out Zoom sentence = object identifier, " is out zoom from ", child OPD ;
```

(* EndRegion: Context management sentences *)
(* EndRegion: OPL document *)
(* EndRegion: OPL EBNF *)

Annex B
(informative)

Guidance for Object-Process Methodology

B.1 Introduction

In view of the rapid development of complex and complicated systems, the need for an intuitive yet formal way of documenting standards for and designs of new systems, or knowledge about existing systems becomes ever more apparent. This need, in turn, requires a solid infrastructure for recording, storing, arranging, and presenting the accumulated knowledge and the creative ideas that build on this knowledge.

Conceptual modelling refers to the practice of representing system-related knowledge. The outcome of this activity is a conceptual model. Conceptual modelling, which usually precedes mathematical and physical modelling, is the primary activity required not only for engineering systems to be understood, designed, and managed, but also for authoring standards that are as complete and as coherent as possible. Modelling is essential and gives rise to model-based systems engineering (MBSE).

Understanding physical, biological, artificial, and social systems and devising standards related to them requires a well-founded, formal, yet intuitive methodology and language that is capable of modelling these complexities in a coherent, straightforward manner. The same modelling paradigm, the heart of the methodology, should serve for both designing new systems and for studying and improving existing systems. The paradigm should apply to artificial as well as natural systems, and faithfully represent physical and informatical things of the modelled domain. Object-Process Methodology (OPM) provides the means to address these aspirations.

NOTE: The remainder of Annex B assumes the reader is familiar with the content of the normative clauses of this International Standard.

B.2 Thing importance OPM principle

Major system-level processes can be as important as, or even more important than objects in the system model. In particular, OPM specifies that the top-level process of an OPM model of a system is the system's function, the value-providing process that embodies the system's purpose and use. Hence, a process must be amenable for modelling independent of any particular set of objects involved in its occurrence.

The relative importance of a thing T in an OPM system model is generally proportional to the highest OPD in the OPD hierarchy where T appears.

B.3 What a new OPD should contain

A good OPD set is readable and easy to follow and comprehend. The following rules of thumb are helpful in deciding when to create a new OPD and ways to keep OPDs as easy to read and grasp as possible:

— the OPD should not stretch over more than one page or one average-size monitor screen;

— the OPD should not contain more than 20–25 things;

— things must not occlude each other, i.e. they are either completely contained within higher-level things, e.g. in case of zooming, or have no overlapping area;

— the diagram should not contain too many links – roughly the same as the number of things;

— a link should not cross the area occupied by a thing; and,.

— the number of links crossing each other should be minimized.

B.4 The element representation OPM principle

An OPM model element appearing in one OPD may appear in any other OPD as the same element. This principle allows the possibility of representing any model element (thing or link) any number of times in as many OPDs as the modeller finds useful. Since a link cannot exists without the things it links, for a link to appear in an OPD, the two things that it links must be present as well

Although a modeller may include any number of things in any OPD, for reasons of clarity and clutter avoidance, it is often highly desirable to include in an OPD only those elements that are needed to grasp a certain aspect or view of the system.

B.5 The multiple thing copies convention

To avoid long and winding links that cross from one side of the OPD to another and clutter it, an OPD may contain multiple copies of the same thing. This multiple thing copies convention complements the element representation OPM principle. Just as an OPM model element appearing in one OPD may appear in any OPD, an OPM element may appear more than once in any OPD. Accordingly, for the sake of avoiding OPD clutter by long, crisscrossing links, a thing may appear at another place in the same OPD using a shorter link. To facilitate recognition of the repetition, the modeller may replace thing symbol by a corresponding duplicate thing symbol – a small object or process slightly showing behind the repeated thing as illustrated in Figure B.1. However, the modeller should use this alternative sparingly as it requires the model reader to notice and keep in mind the longer links that do not appear explicitly in the current OPD context.

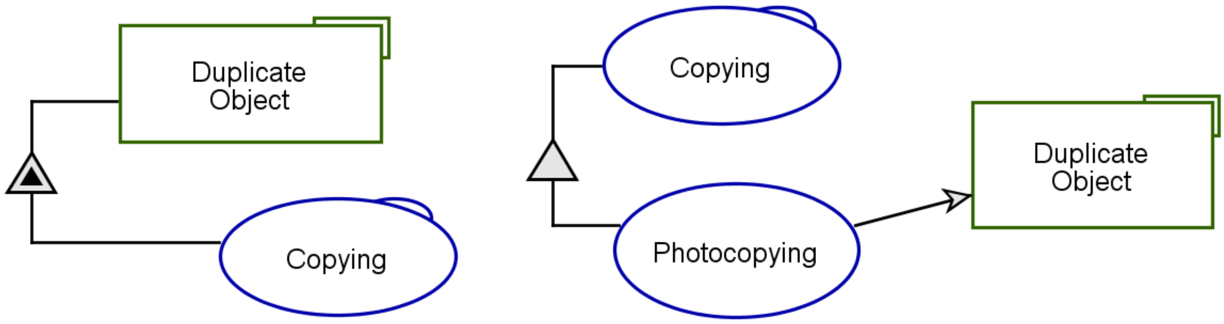


Figure B.1 — Duplicate object and duplicate process symbols

B.6 Naming guidelines

B.6.1 Importance of name selection

Selecting appropriate labelling names for OPM model elements, i.e. the objects, processes, and links, is important because the labels affect the ease of communication to and comprehension of the model by the intended audience and the logical flow and sense-making of the corresponding OPL sentences.

B.6.2 Object naming

A name for an object should be singular. Convert plural names to a singular form. The recommended way to convert an object with several members is to add the word "Set" (usually for inanimate objects) or "Group" (usually for humans) after the singular form.

EXAMPLE 1 "Ingredients" (say, of a cake) becomes "Ingredient Set", while "Customers" becomes "Customer Group".

Because object names must be unique within the system model, the modeller may use the name of a refineable as a prefix for its refine names or may use the name of the refineable as a suffix preceded by "of" after the refine name. Either of these naming schemes allows contextual distinctions when referring to refines with similar semantics.

Object names may be phrases with more than one word, as in Apple Cake or Automobile Crash.

EXAMPLE 2 If a modeller wants **Size** as an attribute of both **Clock Set** and **Watch Set**, then to distinguish between the two **Size** attributes the former may be **Clock Set Size** and the latter **Watch Set Size** or the former may be **Size of Clock Set** and the latter **Size of Watch Set**.

NOTE 1 An implementation of OPM should notify the modeller when an attempt to include an object as a refinee in more than one context occurs so that the modeller may determine the appropriateness of the inclusion.

NOTE 2 An implementation may establish a default syntax to resolve refinee names.

B.6.3 Process naming

A process name is a phrase whose last word should be the gerund form of a verb, i.e. a verb with the "ing" suffix. If there are several choices, such as in Construction vs. Constructing, the latter is preferable.

The following variations for process naming exist:

— the verb version, which is simply the gerund form of the verb, namely verb + ing, as in **Making** or **Responding**;

— the noun-verb version, which is a concatenation of a noun (an OPM object) with the gerund, namely noun + verb + ing, as in **Cake Making** or **Crash Responding**;

— the adjective-verb version, which is a concatenation of an adjective with the gerund form of the verb, namely adjective + verb + ing, as in **Quick Making** or **Automated Responding**; and,

— The adjective-noun-verb version, which is a concatenation of an adjective with a noun with the gerund, namely adjective + noun + verb + ing, as in **Quick Cake Making** or **Automatic Crash Responding**.

In the latter cases, the adjective qualifies the process (the gerund, which is a noun). However, the adjective may also qualify the object (the noun), as in Sweet Cake Making or Fatal Crash Responding.

The name of the function, as well as the names of all OPM processes, should consist of no more than four capitalized words ending with a gerund verb form, e.g. Large City Population Securing.

Because process names must be unique, the modeller may use the name of a refineable as a suffix preceded by "of" after the refine name. The naming scheme allows contextualized distinctions when referring to refines with similar semantics.

B.6.4 State naming

The names of states should reflect the various relevant situations in which their "owning" object can occur at any given point in time. Preferred state names are passive forms of the owning object rather than the gerund form.

EXAMPLE If a **Product** is painted and then inspected, its states should be **painted** and **inspected**, rather than painting and inspecting. **Painting** is the process that changes **Product** from its **unpainted** to its **painted** state, and **Inspecting** changes **Product** from its **painted** state to its **inspected** state. While **Painting** of the **Product** occurs, it has left its **unpainted** state for as long as **Painting** takes place and it is in transition between states and has not yet entered its **painted** state until **Painting** is complete.

B.6.5 Capitalization convention

In OPM the first letter of each word in the name of a thing (object or process) is capitalized, while the name of an object state or a link is not capitalized. This convention helps to produce OPL sentences that are more readable.

Annex C
(informative)

Modelling OPM using OPM

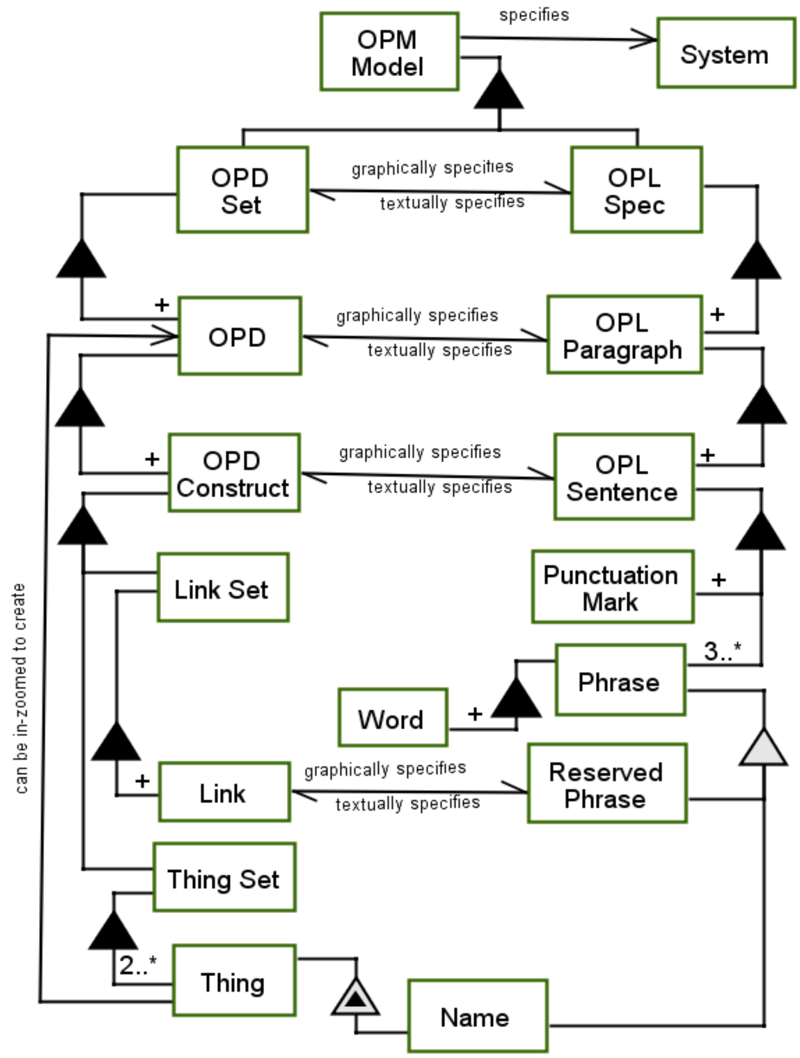
C.1 OPM models of OPM

The OPD in Figure C.1 — OPM model structure

Figure C.1 — OPM model structure C.1 represents aspects of OPM as OPM models. Subclause C.4 elaborates specific elements. Subclause C.5 presents a model relating to the treatment of links during unfolding and in-zooming. Subclause C.6 presents a model for evaluating process invocation, performance, and completion.

This set of sub-clauses expresses OPM as a set of OPD together with the corresponding OPL. For this presentation, the modeller has chosen to limit the model contents to relatively simple OPM usage, i.e. compound links are minimal and there is no attempt to unify the individual OPD into a single OPM model. However, some advanced OPL expressions that limit the redundancy of text and aid in clarifying otherwise distinct but related model facts do occur.

C.2 OPM model structure



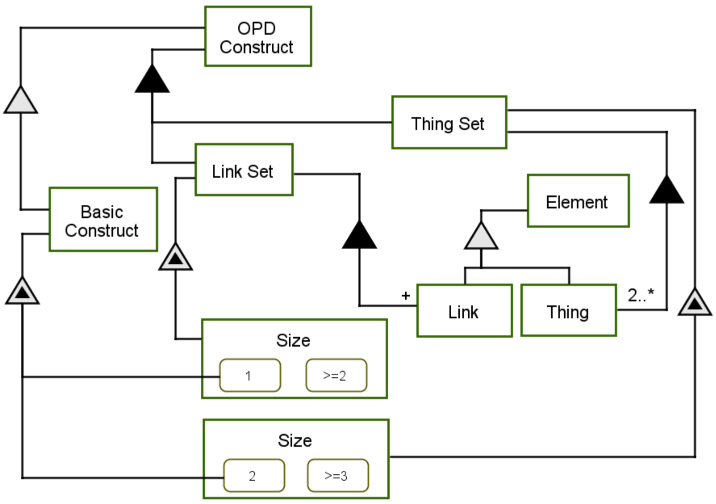
OPM Model specifies **System.**
OPM Model consists of **OPD Set** and **OPL Spec.**
OPL Spec consists of at least one **OPL Paragraph.**
OPD Set consists of at least one **OPD.**
OPD Set graphically specifies OPL Spec.
OPL Spec textually specifies OPD Set.
OPD consists of at least one **OPD Construct.**
OPL Paragraph consists of at least one **OPL Sentence.**
OPD graphically specifies OPL Paragraph.
OPL Paragraph textually specifies OPD.
OPD Construct graphically specifies OPL Sentence.
OPL Sentence textually specifies OPD Construct.
OPD Construct consists of **Thing Set** and **Link Set.**
Thing Set consists of two to many **Things.**
Link Set consists of at least one **Link.**
Thing exhibits **Name.**
OPL Sentence consists of three to many **Phrases** and at least one **Punctuation Mark.**
Phrase consists of at least one **Word.**
OPL Reserved Phrase and **Name** of **Thing** are **Phrases.**
Link graphically specifies Reserved Phrase.
Reserved Phrase textually specifies Link.
Thing can be in-zoomed to create **OPD**

Figure C.1 — OPM model structure

Figure C.1 — OPM model structure**Figure C.1 — OPM model structure**

Figure C.1 — OPM model structure, is a model of the structure of an **OPM model** that depicts the conceptual aspects of OPM as parallel hierarchies of the graphic and textual OPM modalities and their correspondence to produce equivalent model expressions. An **OPD Construct** is the graphical expression of the corresponding textual **OPL Sentence**, which express the same model fact. An **OPD** and its corresponding **OPL Paragraph** are collections of model facts that a modeller places into the same model context.

C.3 OPD Construct model



OPD Construct consists of **Thing Set** and **Link Set.**
Thing and **Link** are **Elements.**
Thing Set consists of **2** to many **Things.**
Link Set consists of at least one **Link.**
Thing Set exhibits **Size** of **Thing Set.**
Link Set exhibits **Size** of **Link Set.**
Size of **Thing Set** can be **2** or **>=3.**
Size of **Link Set** can be **1** or **>=2.**
Basic Construct is an **OPD Construct**.
Basic Construct exhibits **1 Size of Link Set.**
Basic Construct exhibits **2 Size of Thing Set.**

Figure C.2 — Model of OPD Construct and Basic Construct

Figure C.2 — Model of OPD Construct and Basic Construct

, elaborates the **OPD Construct** concept. The purpose of this model is to distinguish **Basic Construct** from another possible **OPD Construct.** A **Basic Construct** is a specialization of **OPD Construct**, which consists of exactly two **Things** connected by exactly one **Link**, The non-basic constructs include, among others, those with link fans or more than two refinees.

EXAMPLE 1 In Figure C.1 — OPM model structure

Figure C.1 — OPM model structure, the two objects **OPM Model** and **OPD Set** together with the aggregation-participation link from the former to the latter constitute a basic construct. The OPL sentence that is equivalent to this basic construct is: **OPM Model** consists of **OPD Set.**

EXAMPLE 2 In Figure C.1 — OPM model structure

Figure C.1 — OPM model structure, the three objects **OPM Model**, and **OPD Set**, and **OPL Spec** together with the aggregation-participation link from **OPM Model** to **OPD Set** and **OPL Spec** constitute a compound construct. The OPL sentence that is equivalent to this basic construct is: **OPM Model** consists of **OPD Set** and **OPL Spec.**

NOTE An object-state link is implicit between an object and each one of its states. Graphically, this link expression occurs by placing the state inside the object rectangle, effectively linking the state with the object. Therefore, an object with two or more states is an **OPD Construct**, and an object with one state is a **Basic Construct**. A stateless object is not a construct at all, as it has not even an implicit link.

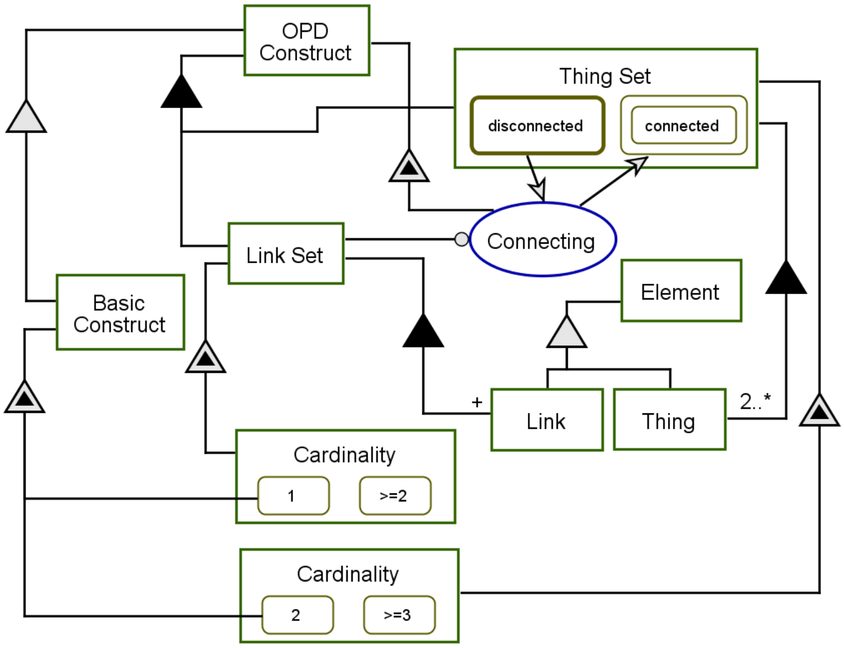
In some situations, the syntax of two constructs combine easily into a compound OPL sentence that reduces redundancy in the text as shown in the next model variation for **OPD Construct**.

A modeller could add a process to the model of Figure C.2 — Model of OPD Construct and Basic Construct

,Figure C.2 — Model of OPD Construct and Basic Construct

to indicate that the **OPD Construct** exhibits **Connecting** as shown in Figure C.3 — OPD Construct and Basic Construct construction

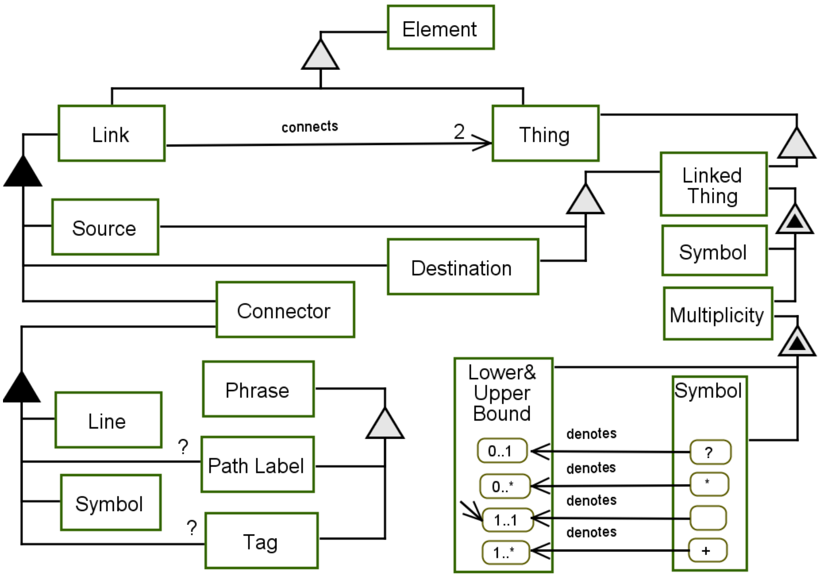
. By adding states **disconnected** and **connected** of **Thing Set**, the purpose of the model thus includes the action of transforming a **disconnected Thing Set** to a **connected Thing Set** using the **Link Set** as an instrument of connection.



OPD Construct consists of **Link Set** and **Thing Set.**
OPD Construct exhibits **Connecting**.
Link Set consists of at least one **Link.**
Link Set exhibits **Cardinality**.
Cardinality of **Link Set** can be **1** or **>=2.**
Thing Set exhibits **Cardinality**.
Thing Set consists of **2** to many **Things**.
Cardinality of **Thing Set** can be **2** or **>=3.**
Link and **Thing** are Elements**.**
Connecting requires **Link Set.**
Connecting changes **Thing Set** from **disconnected** to **connected.**
State **disconnected** of **Thing Set** is **initial.**
State **connected** of **Thing Set** is final**.**
Basic Construct is an **OPD Construct.**
Basic Construct exhibits **1 Cardinality** of **Link Set** and **2 Cardinality** of **Thing Set.**

Figure C.3 — OPD Construct and Basic Construct construction

C.4 OPM Element models

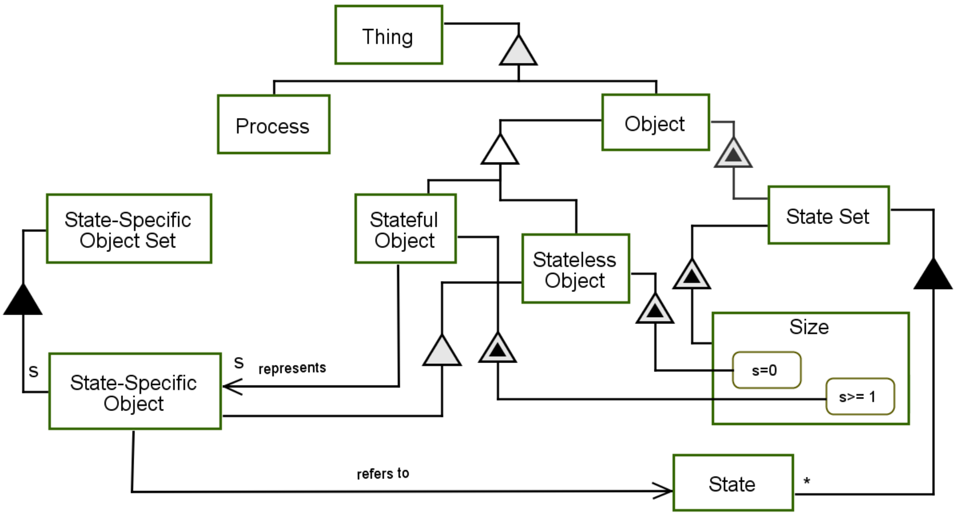


Thing and **Link** are **Elements.**
Link connects 2 Things.
Link consists of **Source, Destination, and Connector.**
Connector consists of **Line, Symbol,** an optional **Tag,** and an optional **Path Label.**
Tag and **Path Label** are **Phrases.**
Source and **Destination** are **Linked Things.**
Linked Thing is a **Thing.**
Linked Thing exhibits **Symbol** and **Multiplicity.**
Multiplicity exhibits **Symbol** and **Lower&Upper Bound.**
Lower&Upper Bound can be **0..1, 0..*, 1..1,** or **1..*.**
Lower&Upper Bound is by default **1..1.**
Symbol of **Multiplicity** can be **?, *, NONE,** or **+.**
? Symbol of **Multiplicity denotes 0..1 Lower&Upper Bound.**
*** Symbol** of **Multiplicity denotes 0..* Lower&Upper Bound.**
NONE Symbol of **Multiplicity denotes 1..1 Lower&Upper Bound.**
+ Symbol of **Multiplicity denotes 1..* Lower&Upper Bound.**

Figure C.4 — OPM model of OPM Element

The model in Figure C.4 — OPM model of OPM Element

, is only valid for basic constructs because **Link connects 2 Things** and not more than two.

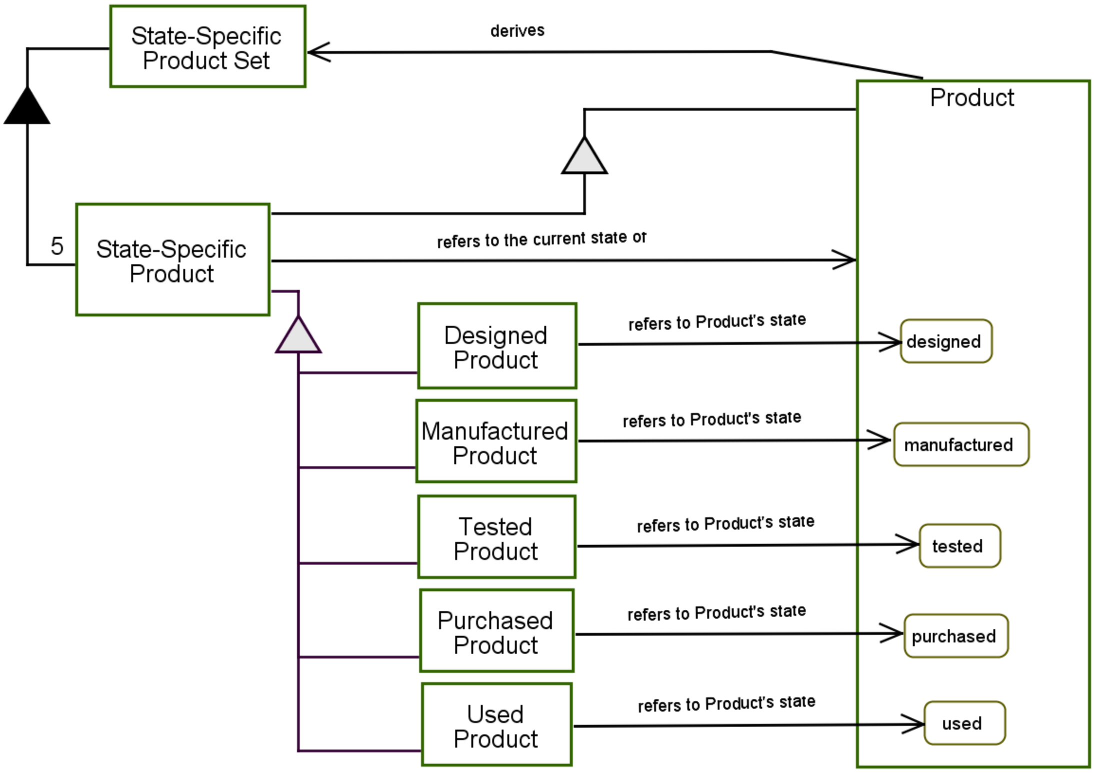


Process and **Object** are **Things.**
Object exhibits **State Set.**
State Set exhibits **Size.**
Cardinality of **State Set** can be **s=0** or **s>= 1.**
State Set consists of optional **States.**
Current State is a **State.**
Stateless Object and **Stateful Object** are **Objects.**
Stateless Object exhibits **s= 0 Size** of **State Set.**
Stateful Object exhibits **s>= 1 Size** of **State Set.**
Stateful Object represents s State-Specific Objects.
State-Specific Object Set consists of **s State-Specific Objects.**
State-Specific Object refers to State.

Figure C.5 — OPM model of Thing

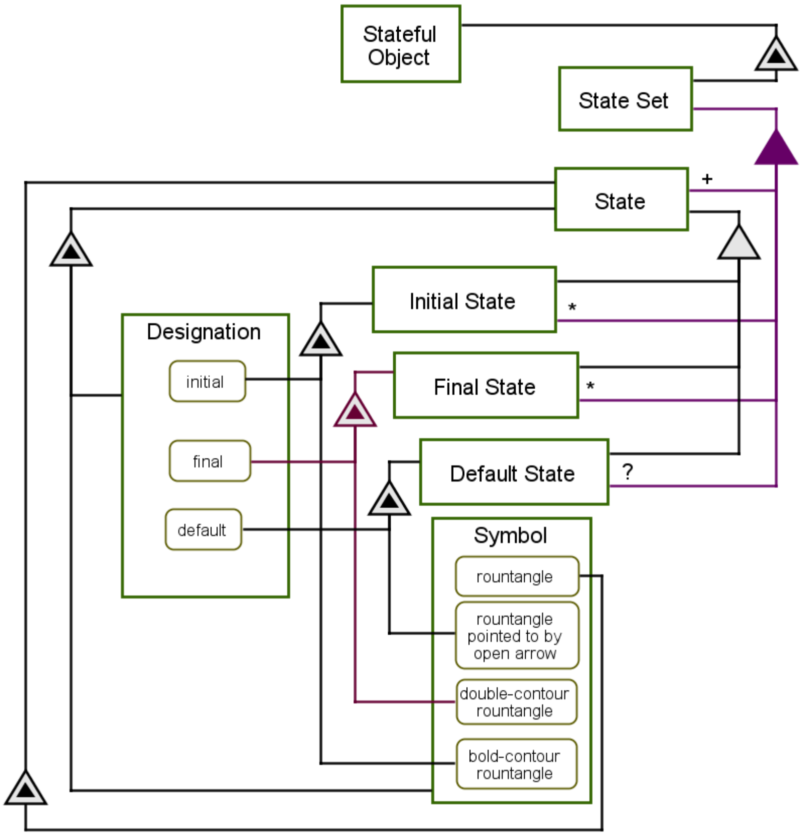
Figure C.5 — OPM model of Thing, is a model for an OPM Thing, showing its specialization into Object and Process. A set of States characterize Object, which can be empty, in a Stateless Object, or non-empty in the case of a Stateful Object. A Stateful Object with s States gives rise to a set of s stateless State-Specific Objects, one for each State. A particular State-Specific Object refers to an object in a specific state. Modelling the concept of State-Specific Object as both an Object and a State enables us to simplify the conceptual model by referring to an object and any one or its states by simply specifying Object.

EXAMPLE In **Error! Reference source not found.**, **Product** is a stateful object with 5 states, from which five istinct specializations of **Product** are derived, each referring to a distinct state of **Product**. Thus, the **State-Specific Product** called **Tested Product** refers to the state **tested** of **Product**. Of course, the same object, **Tested Product**, refers also to **Product** itself, because being a state; “**tested**” has no meaning without reference to the object of which it is a state. This way, there are five **State-Specific Products**, each being a specialization of **Product** and capturing a specific state of **Product**.



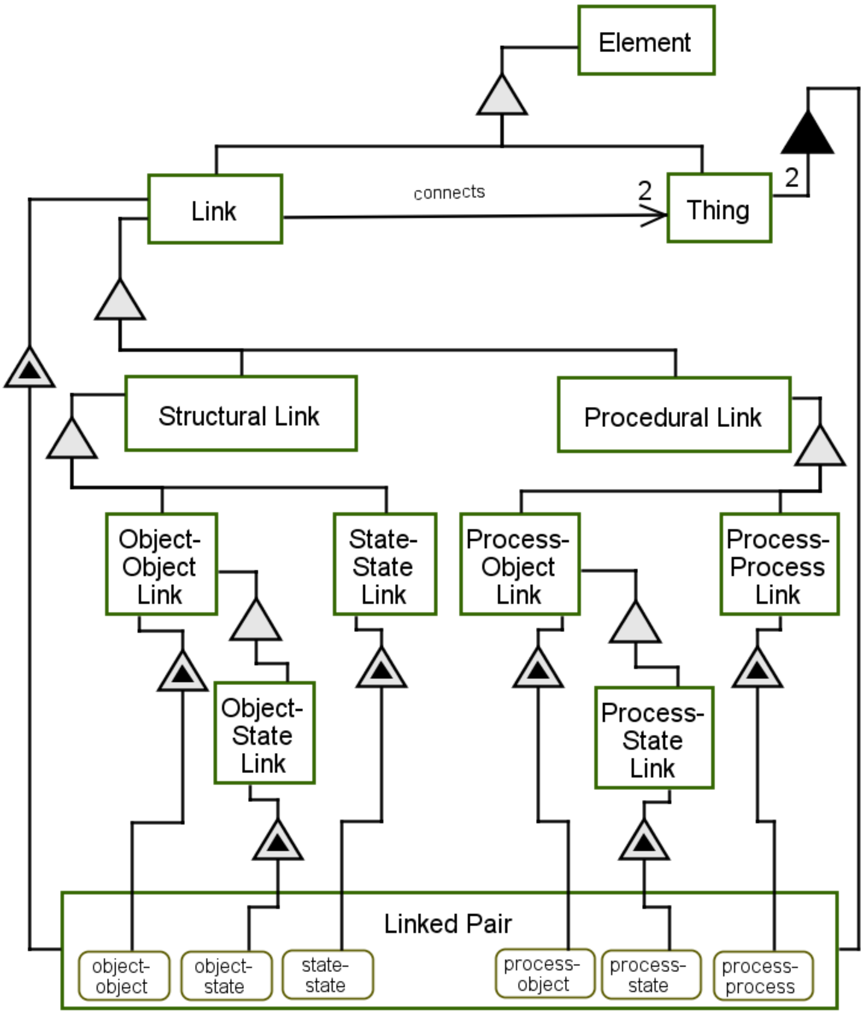
Product can be **designed, manufactured, tested, purchased, or used.**
Product derives State-Specific Product Set.
State-Specific Product Set consists of **5 State-Specific Products.**
State-Specific Product is a **Product.**
State-Specific Product refers to the current state of Product.
Designed Product, Manufactured Product, Tested Product, Purchased Product, and **Used Product** are **State-Specific Products.**
Designed Product refers to Product's state designed.
Manufactured Product refers to Product's state manufactured.
Tested Product refers to Product's state tested.
Purchased Product refers to Product's state purchased.
Used Product refers to Product's state used.

Figure C.6 — Example of state-specific object



Stateful Object exhibits **State Set.**
State Set consists of at least one **State,** optional **Initial States,** optional **Final States,** and an optional **Default State.**
State exhibits **Designation** and **Symbol .**
Designation can be **initial, final,** or **default.**
Initial State, Final State, and Default State are **States.**
Initial State exhibits **initial Designation** and **bold-contour rountangle Symbol** of **State.**
Final State exhibits **final Designation** and **double-contour rountangle Symbol** of **State.**
Default State exhibits **default Designation** and **rountangle pointed to by open arrow Symbol** of **State.**

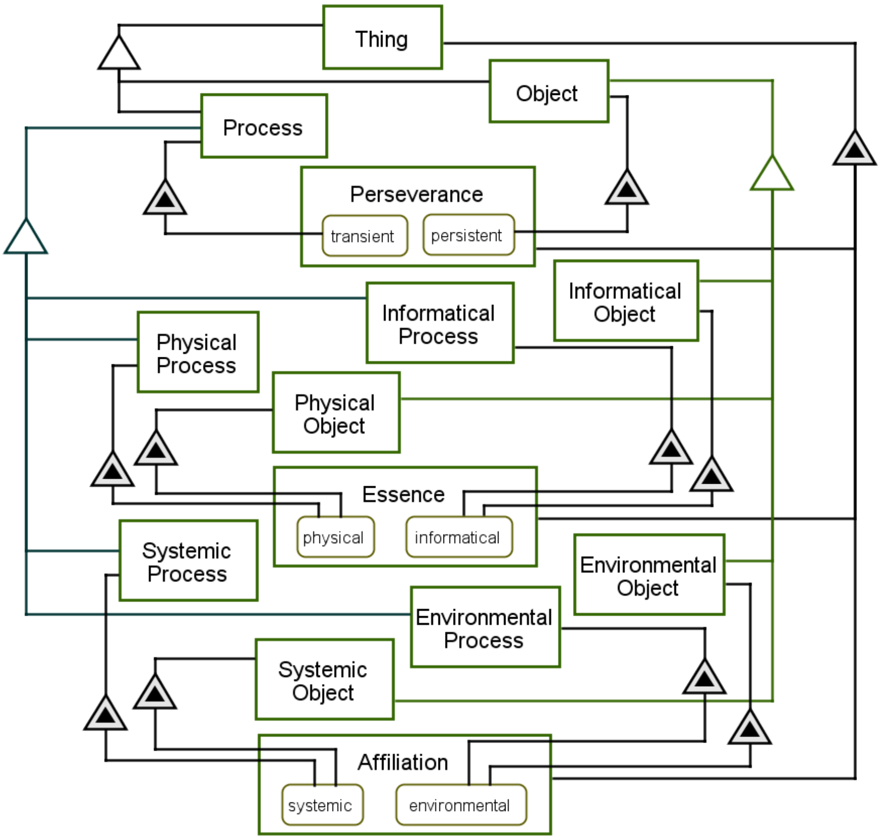
Figure C.7 — OPM model of stateful object and state



Thing and **Link** are **Elements.**
Link connects 2 Things.
Link exhibits **Linked Pair .**
Linked Pair consists of **2 Things.**
Linked Pair can be **object-object, object-state, state-state, process-object, process-state,** or **process-process.**
Structural Link and **Procedural Link** are **Links.**
Object-Object Link and **State-State Link** are **Structural Links.**
Object-State Link is an **Object-Object Link.**
Object-Object Link exhibits **object-object Linked Pair.**
Object-State Link exhibits **object-state Linked Pair.**
State-State Link exhibits **state-state Linked Pair.**
Process-Object Link and **Process-Process Link** are **Procedural Links.**
Process-State Link is a **Process-Object Link.**
Process-Object Link exhibits **process-object Linked Pair.**
Process-State Link exhibits **process-state Linked Pair.**
Process-Process Link exhibits **process- process Linked Pair.**

Figure C.8 — OPM model of links

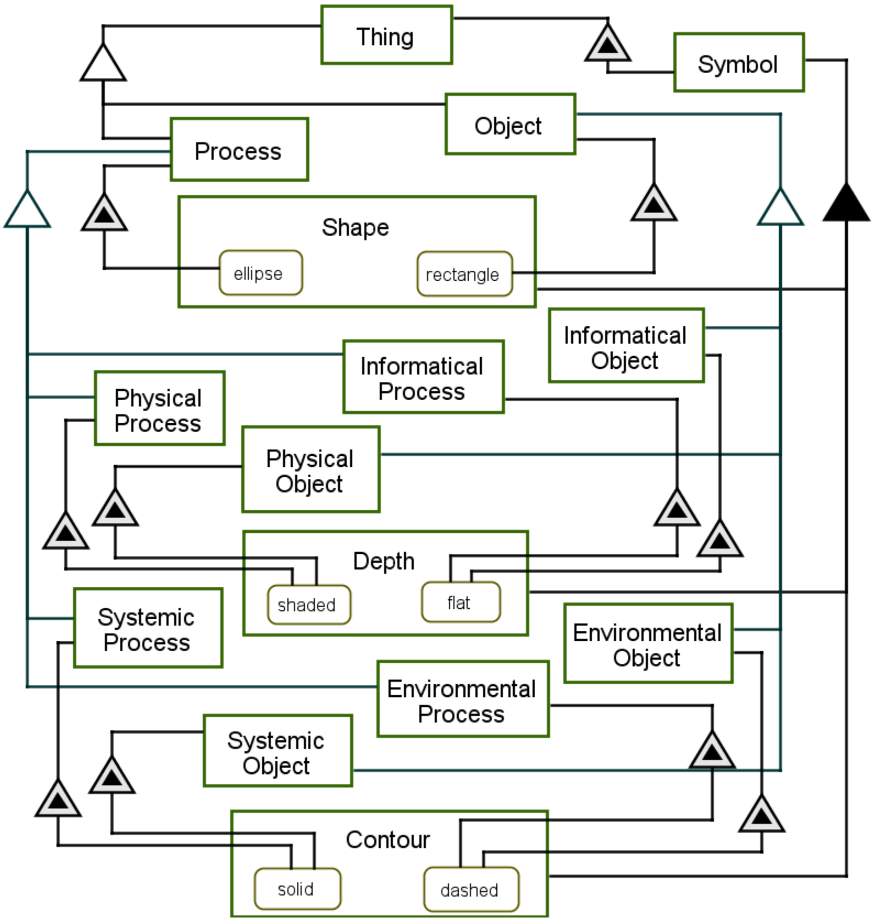
The model in Figure C.8 — OPM model of links is only valid for basic constructs because **Link connects 2 Things** and not more than two.



Thing exhibits **Perseverance, Essence,** and **Affiliation.**
Perseverance can be **transient** or **persistent.**
Essence can be **physical** or **informatical.**
Affiliation can be **systemic** or **environmental.**
Object and **Process** are **Things.**
Process exhibits **transient Perseverance.**
Object exhibits **persistent Perseverance.**
Physical Process, Informatical Process, Systemic Process, and **Environmental Process** are **Processes.**
Physical Object, Informatical Object, Systemic Object, and **Environmental Object** are **Objects.**
Physical Process and **Physical Object** exhibit **physical Essence.**
Informatical Process and **Informatical Object** exhibit **informatical Essence.**
Systemic Process and **Systemic Object** exhibit **systemic Affiliation.**
Environmental Process and **Environmental Object** exhibit **environmental Affiliation.**

Figure C.9 — OPM model of Thing generic properties

Figure C.9 — OPM model of Thing generic properties, depicts **Thing** and its **Perseverance**, **Essence**, and **Affiliation** generic properties modelled as attribute refinees of an exhibition-characterization link. **Perseverance** is the discriminating attribute between **Object** and **Process**. **Essence** is the discriminating attribute between **Physical Object** and **Physical Process** on the one hand, Informatical Object, and **Informatical Process** on the other hand. **Affiliation** is the discriminating attribute between **Systemic Object** and **Systemic Process** on the one hand, Environmental Object, and **Environmental Process** on the other hand.

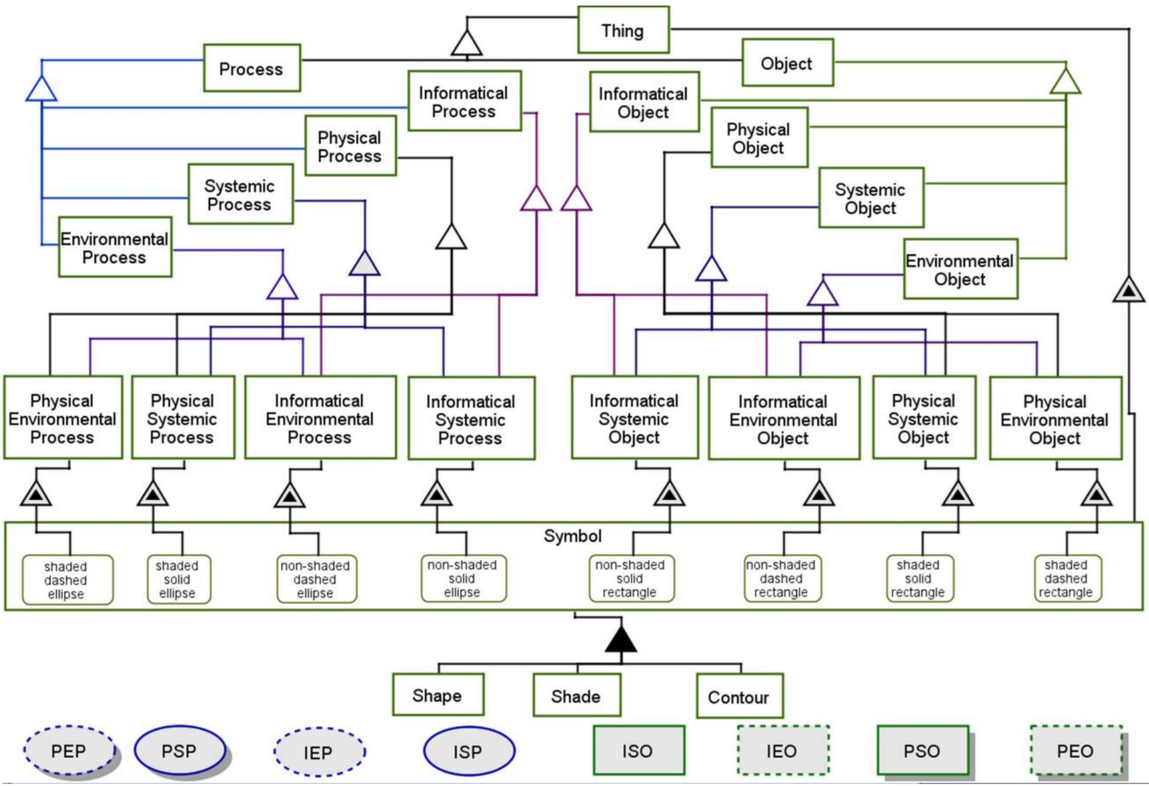


Thing exhibits **Symbol.**
Symbol of **Thing** consists of **Shape, Depth, and Contour.**
Shape can be **ellipse** or **rectangle.**
Depth can be **shaded** or **non- shaded.**
Contour can be **solid** or **dashed.**
Process and **Object** are **Things.**
Process exhibits **ellipse Shape.**
Object exhibits **rectangle Shape.**
Physical Process, Informatical Process, Systemic Process, and **Environmental Process** are **Processes.**
Physical Object, Informatical Object, Systemic Object, and **Environmental Object** are **Objects.**
Physical Process and **Physical Object** exhibit **shaded Depth.**
Informatical Process and **Informatical Object** exhibit **flat Depth.**
Systemic Process and **Systemic Object** exhibit **solid Contour.**
Environmental Process and **Environmental Object** exhibit **dashed Contour.**

Figure C.10 — OPM model of Thing symbolic representation

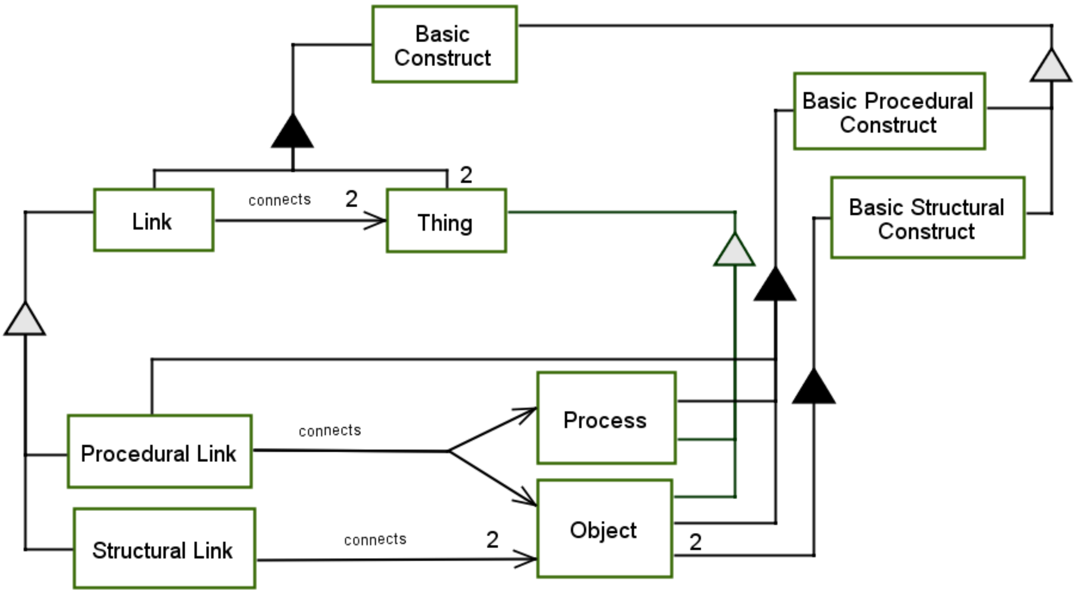
Figure C.10 — OPM model of Thing symbolic representation depicts an OPM model for the graphical representation of OPM things showing a **Symbol** refine attribute and three parts of a **Symbol**: **Shape**, **Depth**, and **Contour**. **Shape** is the part that enables the distinction between **Object** and **Process**. **Depth** is the part that enables the distinction between **Physical Object** and **Physical Process** on the one hand, **Informatical Object** and **Informatical Process** on the other hand. **Contour** is the part that enables the distinction between **Systemic Object** and **Systemic Process** on the one hand, **Environmental Object** and **Environmental Process** on the other hand. Since the states of an object bind to the object, the **Essence** and **Affiliation** associated with a particular **state Object** are the same as that of **Object**.

Figure C.11 — OPM model of the eight Thing symbol representations is a variation of the model in Figure C.10 — OPM model of Thing symbolic representation, in which the three parts of the **Symbol** attribute of **Thing** appear as eight values, one for each of the possible **Thing** configurations. Here, and in several other model figures of this Annex, the actual symbols appear at the bottom of the OPD. In this case, the symbol is below its respective model object and the value of **Symbol** of **Thing**. These eight symbols at the bottom of the OPD are illustrative and thus distinct from the OPD itself. Figure C.11 — OPM model of the eight Thing symbol representations, enhances the Symbol refinee of Figure C.10 — OPM model of Thing symbolic representation by enumerating the eight states of **Symbol**, which are the Cartesian product of the 2x2x2 values of the **Depth**, **Contour**, and **Shape** refinee attributes of **Symbol**.



Thing exhibits **Symbol.**
Symbol of **Thing** consists of **Depth, Contour, and Shape.**
Symbol of **Thing** can be **shaded dashed rectangle, shaded solid ellipse, non-shaded dashed ellipse, non-shaded solid ellipse, non-shaded solid rectangle, non-shaded dashed rectangle, shaded solid rectangle,** or **shaded dashed rectangle.**
Object and **Process** are **Things.**
Physical Process, Informatical Process, Systemic Process, and Environmental Process are **Processes.**
Physical Object, Informatical Object, Systemic Object, and **Environmental Object** are **Objects.**
Physical Systemic Process is a **Physical Process** and a **Systemic Process.**
Physical Systemic Process exhibits **shaded solid ellipse Symbol** of **Thing.**
Physical Environmental Process is a **Physical Process** and an **Environmental Process.**
Physical Environmental Process exhibits **shaded dashed ellipse Symbol** of **Thing.**
Informatical Environmental Process is an **Informatical Process** and an **Environmental Process.**
Informatical Environmental Process exhibits **non-shaded dashed ellipse Symbol** of **Thing.**
Informatical Systemic Process is an **Informatical Process** and a **Systemic Process.**
Informatical Systemic Process exhibits **non-shaded solid ellipse Symbol** of **Thing.**
Physical Environmental Object is a **Physical Object** and an **Environmental Object.**
Physical Environmental Object exhibits **shaded dashed rectangle Symbol** of **Thing.**
Physical Systemic Object is a **Physical Object** and a **Systemic Object.**
Physical Systemic Object exhibits **shaded solid rectangle Symbol** of **Thing.**
Informatical Environmental Object is an **Informatical Object** and an **Environmental Object.**
Informatical Environmental Object exhibits **non-shaded dashed rectangle Symbol** of **Thing.**
Informatical Systemic Object is an **Informatical Object** and a **Systemic Object.**
Informatical Systemic Object exhibits **non-shaded solid rectangle Symbol** of **Thing.**
Symbol of **Thing** consists of **Depth, Contour** and **Shape.**

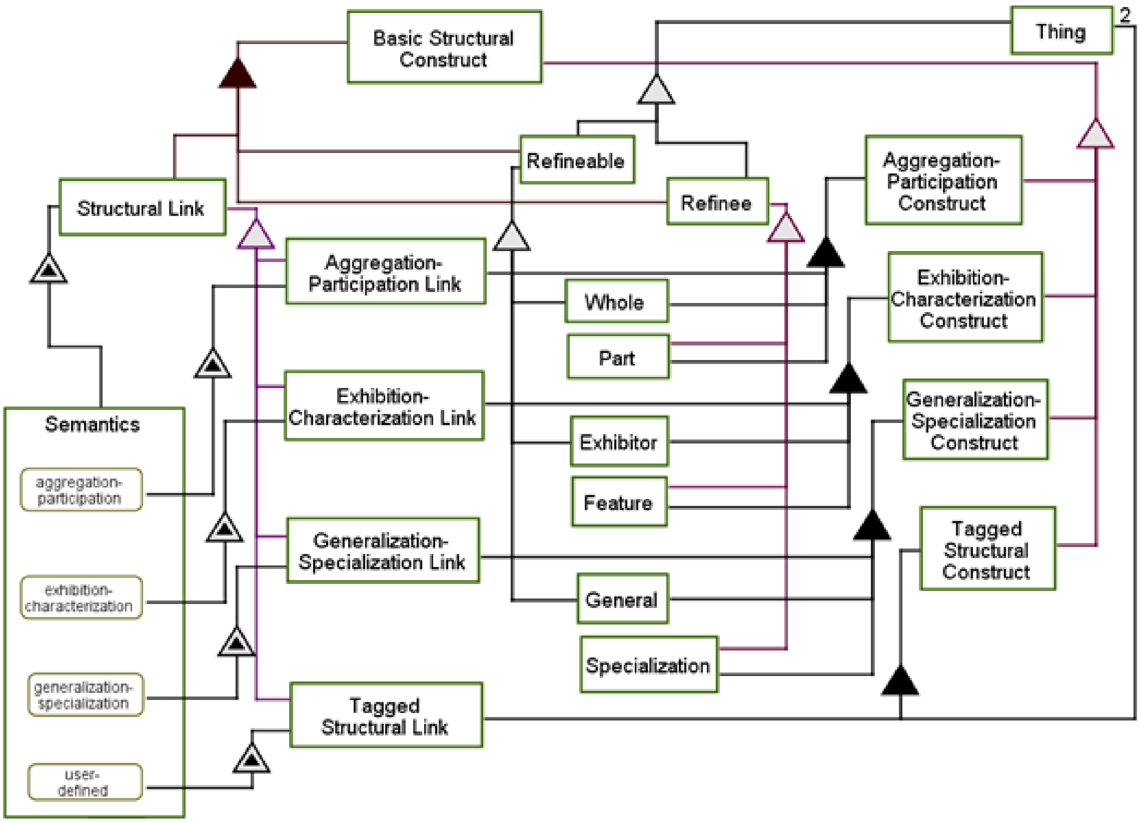
Figure C.11 — OPM model of the eight Thing symbol representations



Basic Construct consists of **Link** and **2 Things.**
Link connects 2 Things.
Structural Link and **Procedural Link** are **Links.**
Basic Structural Construct and **Basic Procedural Construct** are **Basic Constructs.**
Basic Structural Construct consists of **Structural Link** and **2 Objects.**
Basic Procedural Construct consists of **Procedural Link, Object,** and **Process.**
Structural Link connects **2 Objects.**
Procedural Link connects **a Process** and **an Object.**

Figure C.12 — Basic Construct elaboration

The model in Figure C.12 — Basic Construct elaboration is only valid for basic constructs because **Link connects 2 Things** and not more than two.



Basic Structural Construct consists of **Refineable, Refinee,** and **Structural Link.**

Refineable and **Refinee** are **Things.**

Whole, Exhibitor, General, and **Class** are **Refineables.**

Part, Feature, Specialization, and **Instance** are **Refinees.**

Structural Link exhibits **Semantics.**

Semantics can be **aggregation-participation, exhibition-characterization, generalization-specialization, classification-instantiation,** or **user-defined.**

Aggregation-Participation Link, Exhibition-Characterization Link, Generalization-Specialization Link, Classification-Instantiation Link, and **Tagged Structural Link** are **Structural Links.**

Aggregation-Participation Link exhibits **aggregation-participation Semantics.**

Exhibition-Characterization Link exhibits **exhibition-characterization Semantics.**

Generalization-Specialization Link exhibits **generalization-specialization Semantics.**

Classification-Instantiation exhibits **classification-instantiation Semantics.**

Tagged Structural Link exhibits **user-defined Semantics.**

Aggregation- Participation Construct, Exhibition-Characterization Construct, Generalization-Specialization Construct, Classification-Instantiation Construct and **Tagged Structural Construct** are **Basic Structural Constructs.**

Aggregation-Participation Construct consists of **Aggregation-Participation Link, Whole,** and **Part.**

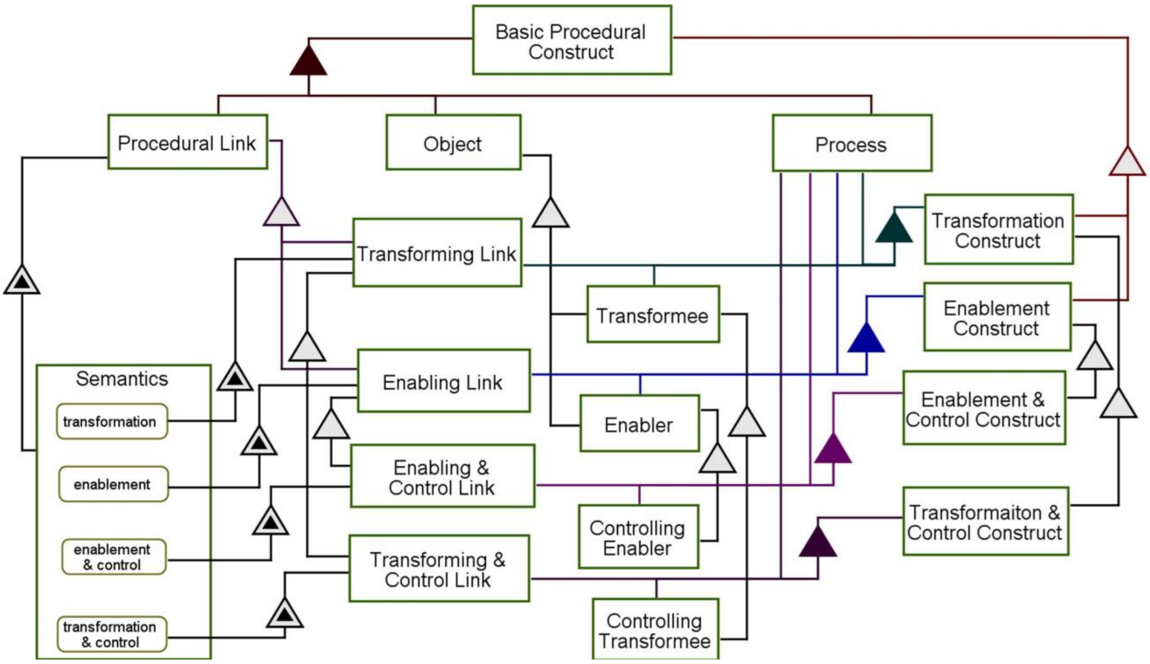
Exhibition- Characterization Construct consists of **Exhibition- Characterization Link, Exhibitor,** and **Feature.**

Generalization- Specialization Construct consists of **Generalization- Specialization Link, General,** and **Specialization.**

Classification-Instantiation Construct consists of **Classification-Instantiation Link, Class, and Instance.**

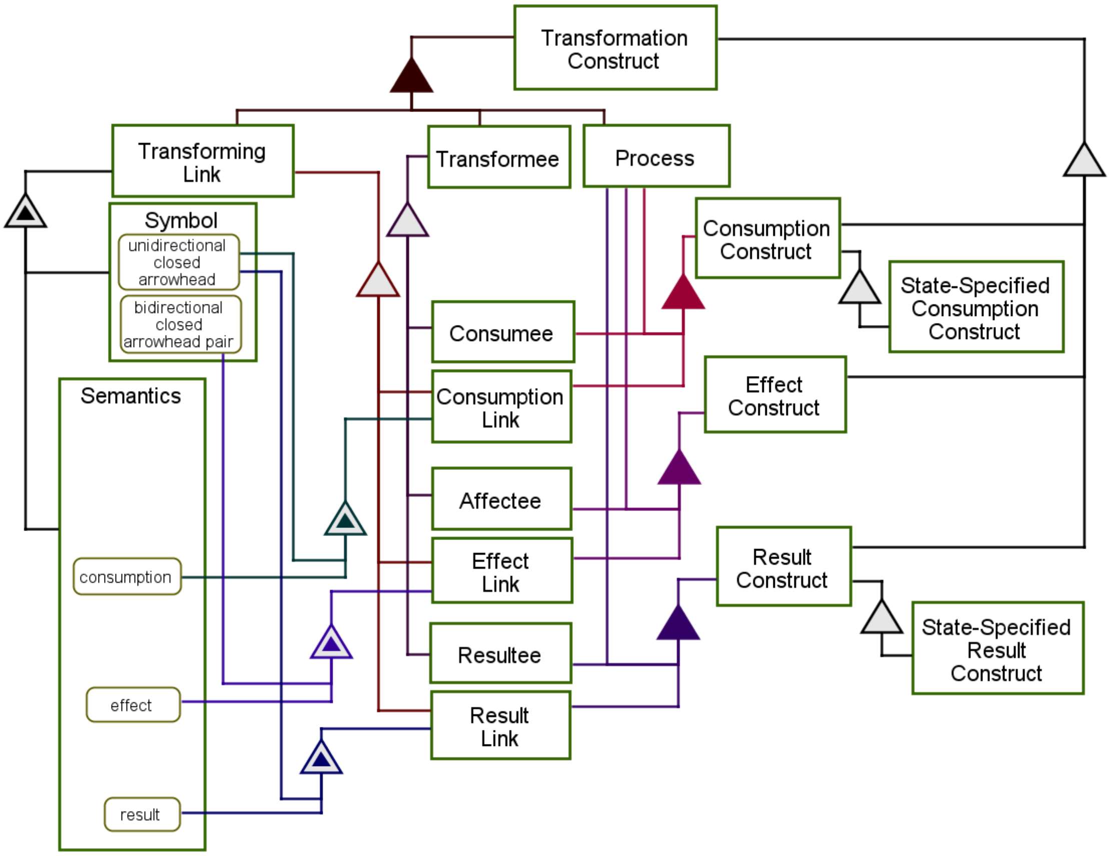
Tagged Structural Construct consists of **Tagged Structural Link** and **2 Things.**

Figure C.13 — OPM model of Basic Structural Construct



Basic Procedural Construct consists of **Object, Process,** and **Procedural Link.**
Procedural Link exhibits **Semantics.**
Semantics of **Procedural Link** can be **transformation, enablement, transformation & control,** and **enablement & control.**
Transformee and **Enabler** are **Objects.**
Controlling Transformee is a **Transformee.**
Controlling Enabler is an **Enabler.**
Transforming Link and **Enabling Link** are **Procedural Links.**
Transforming & Control Link is a **Transforming Link.**
Enabling & Control Link is an **Enabling Link.**
Transforming Link exhibits **transformation Semantics** of **Procedural Link.**
Enabling Link exhibits **enablement Semantics** of **Procedural Link.**
Transforming & Control Link exhibits **transformation & control Semantics** of **Procedural Link.**
Enabling & Control Link exhibits **enablement & control Semantics** of **Procedural Link.**
Transformation Construct and **Enablement Construct** are **Basic Procedural Constructs.**
Transformation Construct consists of **Transforming Link, Transformee,** and **Process.**
Enablement Construct consists of **Enablement Link, Enabler,** and **Process.**
Transformation & Control Construct is a **Transformation Construct.**
Enablement & Control Construct is an **Enablement Construct.**
Transformation & Control Construct consists of **Transforming & Control Link, Controlling Transformee,** and **Process.**
Enablement & Control Construct consists of **Enablement & Control Link, Controlling Enabler,** and **Process.**

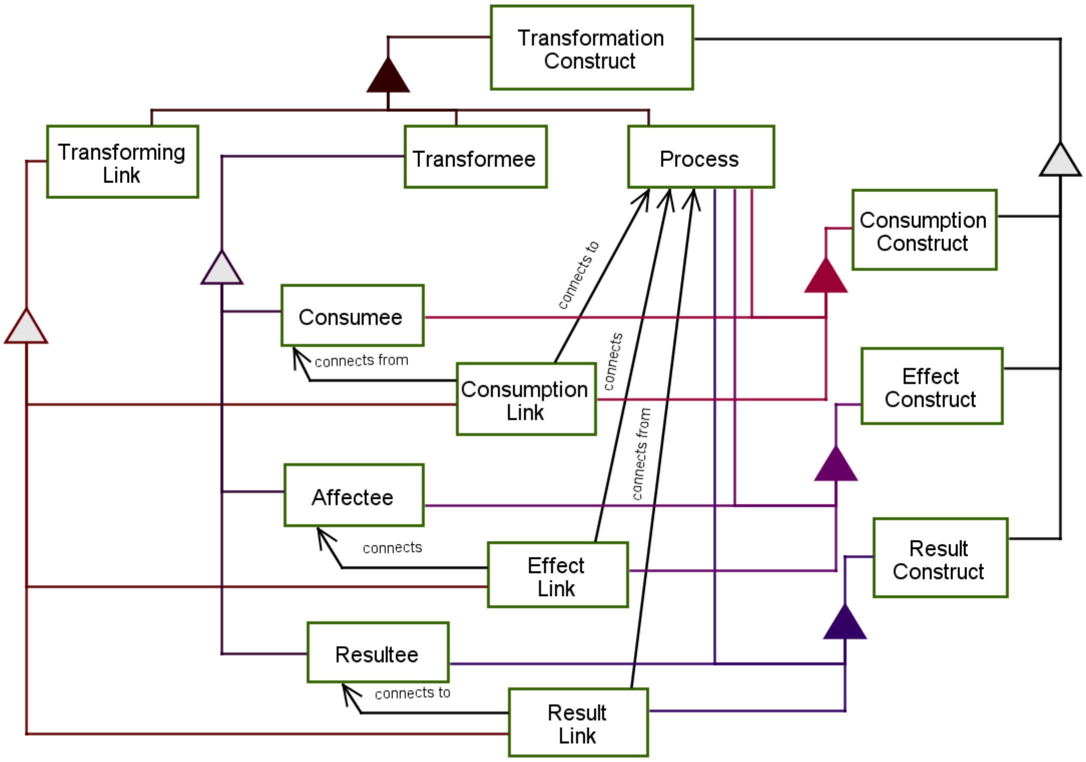
Figure C.14 — OPM model of Basic Procedural Construct



Transformation Construct consists of **Transformee, Process,** and **Transforming Link.**
Transforming Link exhibits **Symbol** and **Semantics.**
Symbol of **Transforming Link** can be **unidirectional closed arrowhead** or **bidirectional closed arrowhead pair.**
Semantics of **Transforming Link** can be **consumption, effect, or result.**
Consumption Link, Effect Link, and **Result Link** are **Transforming Links.**
Effect Link exhibits **effect Semantics** of **Transforming.**
Result Link exhibits **result Semantics** of **Transforming.**
Consumee, Affectee, and **Resultee** are **Transformees.**
Consumption Construct, Result Construct, and **Effect Construct** are **Transformation Constructs.**
Consumption Construct consists of **Consumption Link, Process,** and **Consumee.**
Effect Construct consists of **Effect Link, Process,** and **Affectee.**
Result Construct consists of **Result Link, Process,** and **Resultee.**
Consumption Link exhibits **unidirectional closed arrowhead Symbol** of **Transforming Link** and **consumption Semantics** of **Transforming Link.**
Effect Link exhibits **bidirectional closed arrowhead consumption pair** of **Transforming Link** and **effect Semantics** of **Transforming Link.**
Result Link exhibits **unidirectional closed arrowhead Symbol** of **Transforming Link** and **result Semantics** of **Transforming Link.**
State-Specified Consumption Construct is a **Consumption Construct.**
State-Specified Result Construct is a **Result Construct.**

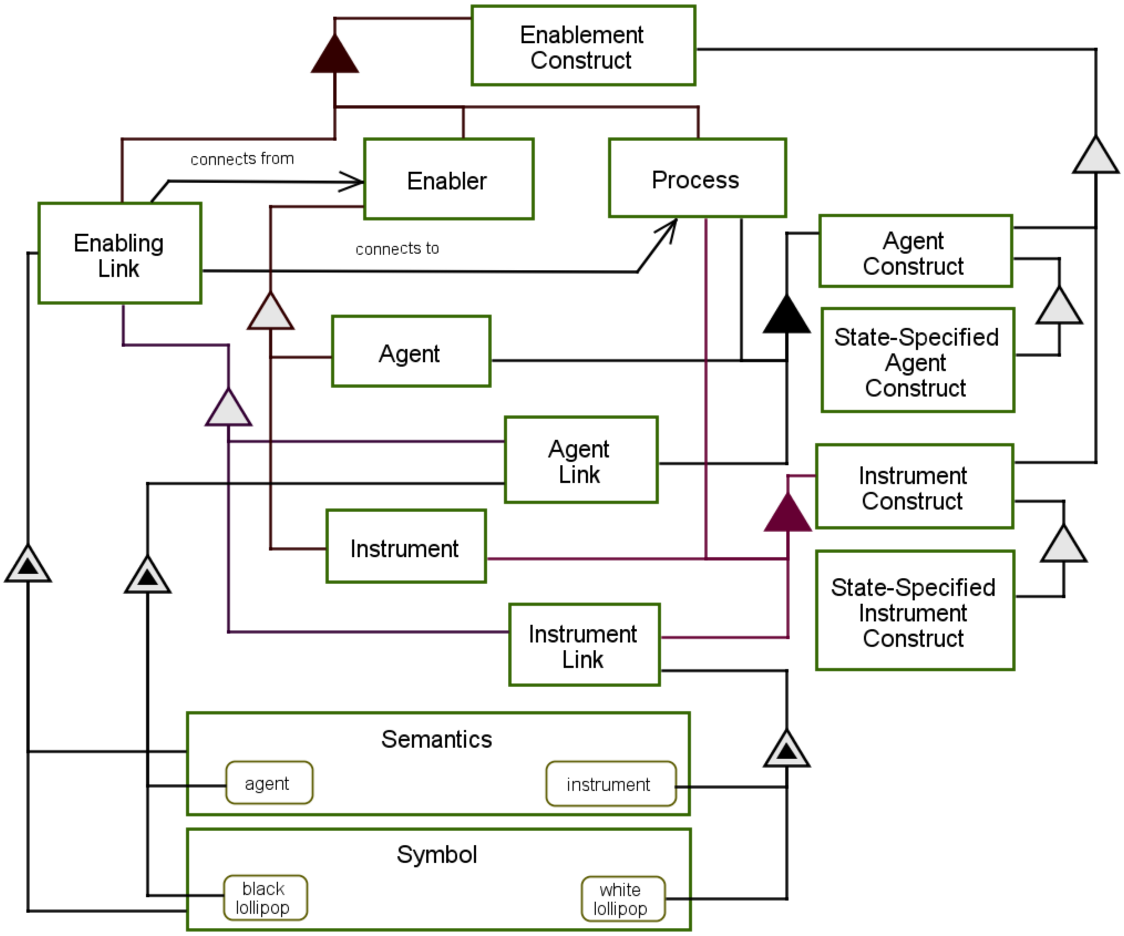
Figure C.15 — OPM model of Transformation Construct

Figure C.16 — OPM model of Transformation Construct link directionality complements Figure C.15 — OPM model of Transformation Construct by adding information about the directionality of the arrowhead symbols that connect an object with the process. Adding this information to Figure C.15 — OPM model of Transformation Construct could clutter the model figure and make it more difficult to comprehend.



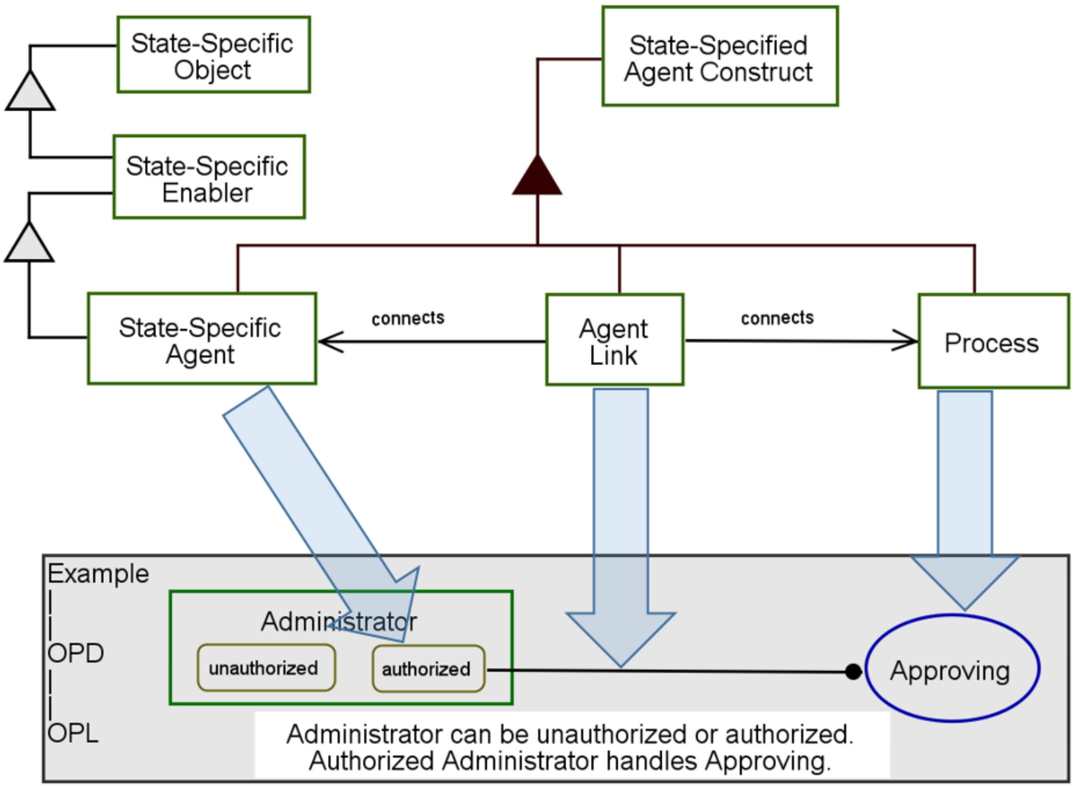
Transformation Construct consists of **Transformee, Process,** and **Transforming Link.**
Consumption Link, Effect Link, and **Result Link** are **Transforming Links.**
Consumption Construct, Result Construct, and **Effect Construct** are **Transformation Constructs.**
Consumption Construct consists of **Consumption Link, Process,** and **Consumee.**
Effect Construct consists of **Effect Link, Process,** and **Affectee.**
Result Construct consists of **Result Link, Process**, and **Resultee.**
Consumption Link connects from **Consumee.**
Consumption Link connects to **Process.**
Effect Link connects **Affectee** and **Process.**
Result Link connects to **Resultee.**
Result Link connects from **Process.**

Figure C.16 — OPM model of Transformation Construct link directionality



Enablement Construct consists of **Enabler, Process**, and **Enabling Link.**
Enabling Link exhibits **Semantics** and **Symbol.**
Enabling Link connects from Enabler.
Enabling Link connects to Process.
Semantics of **Enabling Link** can be **Agent** or **Instrument.**
Symbol of **Enabling Link** can be **black lollipop** or **white lollipop.**
Agent and **Instrument** are **Enablers.**
Agent Link and **Instrument Link** are **Enabling Links.**
Agent Link exhibits **agent Semantics** of **Enabling Link** and **black lollipop Symbol** of **Enabling Link.**
Instrument Link exhibits **instrument Semantics** of **Enabling Link**
and **white lollipop Symbol** of **Enabling Link.**
Agent Construct and **Instrument Construct** are **Enablement Constructs.**
Agent Construct consists of **Agent, Process**, and **Agent Link.**
Instrument Construct consists of **Instrument, Process**, and **Instrument Link.**
State-Specified Agent Construct is an **Agent Construct.**
State-Specified Instrument Construct is an **Instrument Construct.**

Figure C.17 — OPM model of Basic Enablement Construct



State-Specified Agent Construct consists of **State-Specified Agent, Process**, and **Agent Link.**
State-Specified Agent is a **State-Specified Enabler.**
State-Specified Enabler is a **State-Specified Object.**
Agent Link connects State-Specified Agent and Process.

Figure C.18 — OPM model of state-specified agent construct with mapped example

Figure C.18 — OPM model of state-specified agent construct with mapped example depicts two OPM models with the top of the figure expressing essential associations for a State-Specified Agent Construct and the bottom of the figure expressing a corresponding model construct. The former provides a metamodel for the latter. The broad arrows map the conceptual parts of the construct to the OPD symbols of the example. Below the OPD in the example is the corresponding OPL.

For instructional purposes, similar mapping figures may express the correspondence between models of OPM construct conceptual models and corresponding OPM models in application.

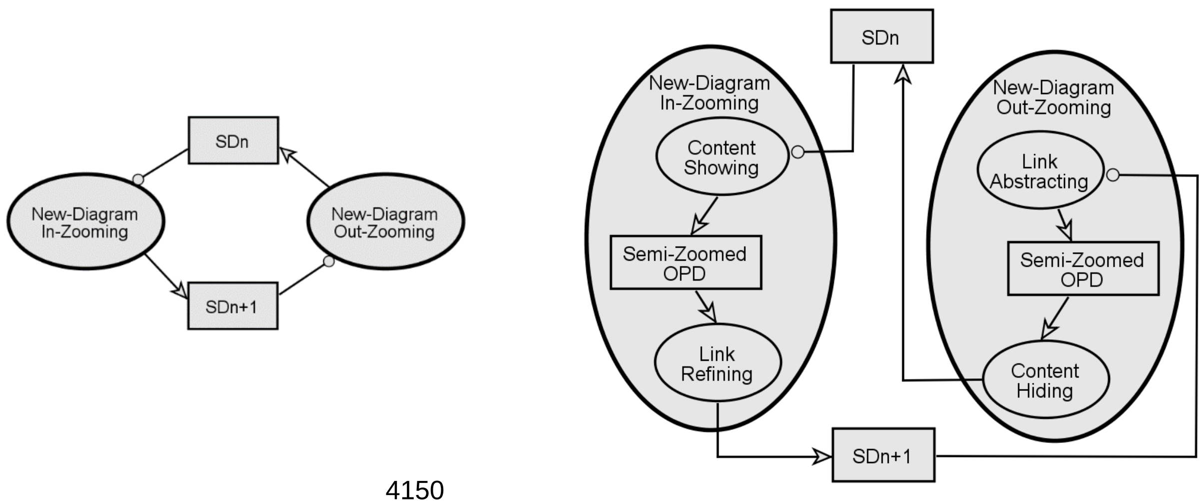
C.5 In-zooming and out-zooming models

C.5.1 The in-zooming and out-zooming mechanisms

Both new-diagram in-zooming and new-diagram out-zooming create a new OPD context from an existing OPD context. New-diagram in-zooming starts with an OPD of relatively less details and adds elaboration or refinement as a descendant OPD that applies to a specific thing in the less detailed OPD. New-diagram out-zooming starts with an OPD of relatively more details and removes elaboration or refinement to produce a less detailed, more abstract thing in an ancestor context.

New-diagram in-zooming elaborates a refineable present in an existing OPD, say SDn, by creating a new OPD, SDn+1, which elaborates the refineable by adding subprocesses associated objects, and relevant links. The new-diagram in-zooming and in new-diagram out-zooming processes are inverse operations.

Figure C.19 — New-Diagram In-Zooming and New-Diagram Out-Zooming models depicts the **New-Diagram In-Zooming** and **New-Diagram Out-Zooming** processes. The model on the right uses in-diagram in-zooming of the model on the left to elaborate the two processes, one for creating a new-diagram in-zoomed context and one for creating a new-diagram out-zoomed context. **New-Diagram In-Zooming** begins with **Content Showing**, followed by **Link Refining**. **New-Diagram Out-Zooming** begins with **Link Abstracting**, the inverse process of **Link Refining**, followed by **Content Hiding**, the inverse process of **Content Showing**.



New-Diagram In-Zooming requires **SDn.**
New-Diagram In-Zooming yields **SDn+1.**
New-Diagram In-Zooming yields **SDn+1.**
New-Diagram Out-Zooming requires **SDn+1.**

New-Diagram In-Zooming zooms into **Content Showing**
and **Link Refining** in that sequence, as well as **Semi-Zoomed OPD.**
Content Showing requires **SDn.**
Content Showing yields **Semi-Zoomed OPD.**
Link Refining consumes **Semi-Zoomed OPD.**
Link Refining yields **SDn+1.**
New-Diagram Out-Zooming zooms into **Link Abstracting**
and **Content Hiding** in that sequence,
as well as **Semi-Zoomed OPD.**
Link Abstracting requires **SDn+1.**
Link Abstracting yields **Semi-Zoomed OPD.**
Content Hiding consumes **Semi-Zoomed OPD.**
Content Hiding yields **SDn.**

Figure C.19 — New-Diagram In-Zooming and New-Diagram Out-Zooming models

Semi-Zoomed OPD is an interim object created and subsequently consumed during **New Diagram In-Zooming** or **New-Diagram Out-Zooming**. **Semi-Zoomed OPD** appears only within the contexts of **New-Diagram In-Zooming** and **New-Diagram Out-Zooming**.

Figure C.20 — New-Diagram In-Zooming and New-Diagram Out-Zooming elaboration shows **New-Diagram In-Zooming** and **New-Diagram Out-Zooming** with unfolding of **SDn**, **SDn+1**, and **Semi-zoomed OPD** from

Figure C.19 — New-Diagram In-Zooming and New-Diagram Out-Zooming models. **New-Diagram In-Zooming** and **New-Diagram Out-Zooming** operate on a particular instance of **SDn** shown at the middle top of Figure C.20 — New-Diagram In-Zooming and New-Diagram Out-Zooming elaboration, where the **SDn** detail is one of many possibilities. In this case, **SDn** includes **P**, which is the refineable process, as well as four objects connected to **P** with different kinds of links: the consumee **C**, the agent **A**, the instrument **D**, and the resultee **B**.

The in-diagram in-zooming of **Semi-Zoomed OPD** makes clear that it is an interim representation created and consumed during **New Diagram In-Zooming** as well as during **New Diagram Out-Zooming**. The **Semi-Zoomed OPD** is the same in both situations.

Content Showing is the first of the two **New-Diagram In-Zooming** subprocesses. During **Content Showing**, the boundary of **P** expands to make room for showing its content—the model subprocesses **P1**, **P2**, and **P3**, as well as the interim model object **BP**. The result of **Content Showing** is the unfolding of object **Semi-Zoomed OPD**. As an interim object, recognizable only in the context of **New-Diagram In-Zooming**, the second subprocess, **Link Refining**, consumes it while creating **SDn+1**. During **Link Refining**, the procedural links attached to the contour of **P** migrate to the appropriate subprocesses as determined by the modeller. Thus, since **P1** consumes **C**, the consumption link arrowhead migrates from **P** to **P1**. The agent **A** handles both **P1** and **P2**, so in **SDn+1** two agent links, one to **P1** and the other to **P2**, replace the single one in **SDn** from **A** to **P**. **P3** requires **D**, so the instrument link moves from **P** to **P3**. Finally, since **BP** results from **P1** and **P3** consumes it, the corresponding result and consumption links are added, making **BP** an internal object of **P**, an object that is only recognizable within the context of **P**, like **P1**, **P2**, and **P3**. Notice that **BP** is to **P** as **Semi-Zoomed OPD** is to **New-Diagram In-Zooming**.

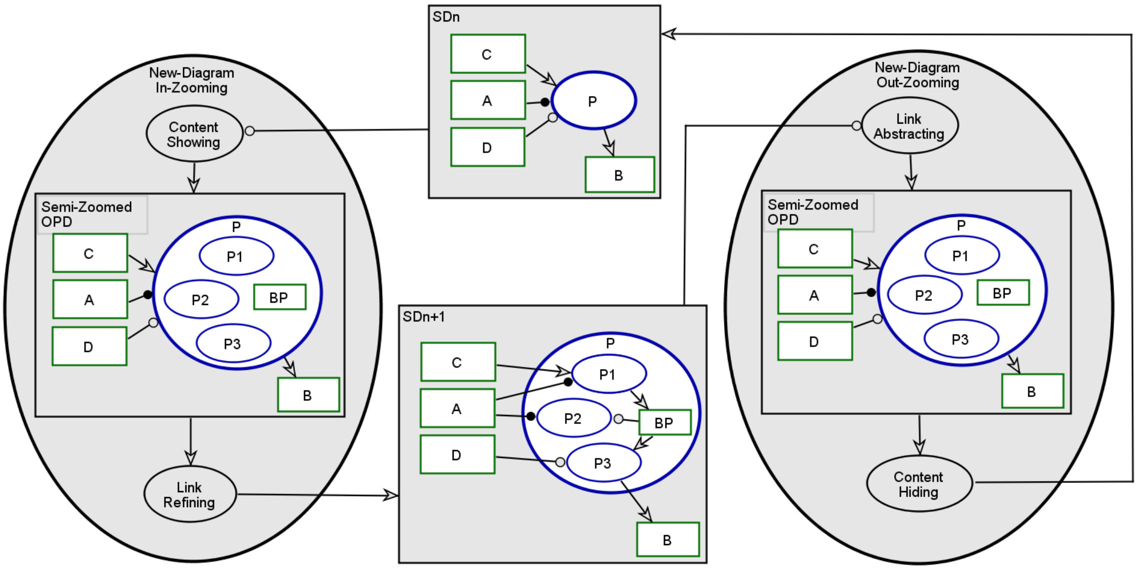


Figure C.20 — New-Diagram In-Zooming and New-Diagram Out-Zooming elaboration

C.5.2 Simplifying an OPD

In-diagram out-zooming can combine with new-diagram in-zooming to simplify an already-modelled OPD that the modeller deems overly complicated. In-diagram out-zooming followed by new-diagram in-zooming is an option when the modeller realizes that the current OPD is overloaded with details. In-diagram out-zooming reduces the cognitive load necessary to understand the complicated OPD at the expense of adding a new OPD to the OPD set, which is the result of the subsequent new-diagram in-zooming.

Figure C.21 — Simplifying an OPD, demonstrates in-diagram out-zooming followed by new-diagram out-zooming. On the left is the original OPD Set with three OPDs: **SD**, **SD1** and **SD1.1**. The modeller deems **SD1** overly complicated. To ease the complication, as shown in the middle, the modeller selects **P1**, **P2**, and **P3**, along with **BP** for replacement by **P123** using new-diagram out-zooming. On the right is the new OPD Set with four OPDs renumbered to reflect the new hierarchy. The new **SD1** is less complicated than the original **SD1**, having five fewer elements (three processes, one object, and two links removed; one process—**P123**—added). **P123** undergoes new-diagram out-zooming in the new **SD1.1**, and this new OPD is inserted into the process hierarchy, pushing the old **SD1.1** to become the new **SD1.1.1**.

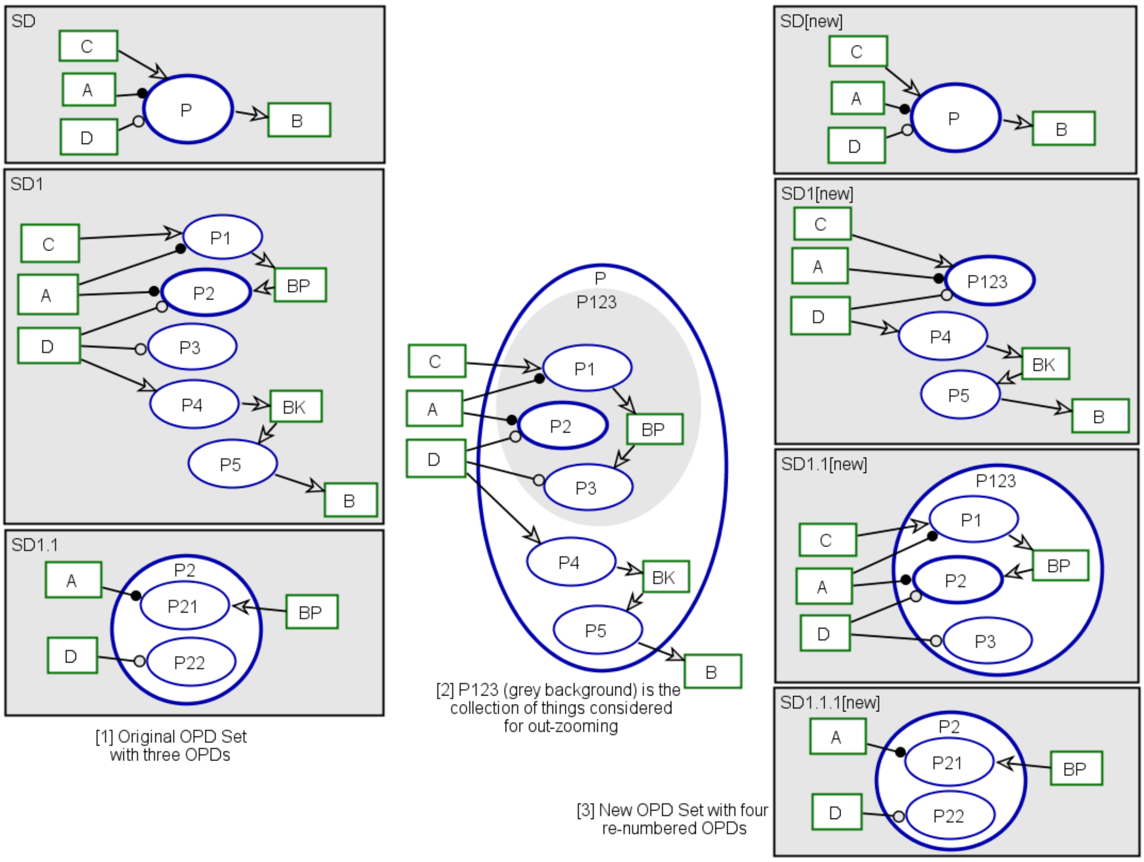


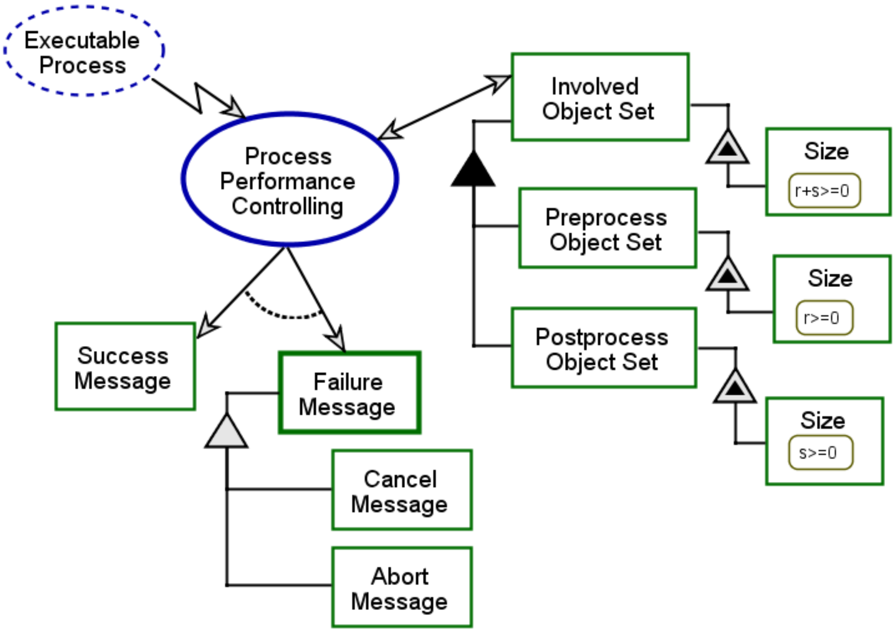
Figure C.21 — Simplifying an OPD

In-diagram out-zooming begins by selecting the set TO of things to out-zoom in the currently complicated OPD for in-zooming in a new OPD. Assuming a new single process, PA, replaces the TO set, each procedural link that extends to a member of TO needs to connect to the new process, PA, and to an object that is not a member of the set TO. PA is a new abstract process that replaces the members of TO and becomes a new model element. PA becomes in-zoomed in a new OPD and the OPD set labelling needs to reflect the new OPD hierarchy.

In the middle of Figure C.21 — Simplifying an OPD the processes **P1**, **P2**, and **P3**, along with the object **BP** are the four members of TO, which are surrounded by **P123**. The consequence of creating **P123** is the disappearance of the four members of TO from the new SD1. Each link that crosses the grey-white boundary of the middle graphic now connects to the boundary of P123 in the new SD1. The objects connecting to the boundary of P123 in the new SD1 then connect to the appropriate subprocesses in the new SD1.1 The object **BK** cannot be a member of TO because if **BK** occurs in **P123** its links create two procedural links connecting two processes directly, **P4** to **P123** and **P123** to **P5**. OPM does not define the semantics of these links and the model would violate the specification that every procedural link (except the invocation and time exception links) connects an object to a process.

C.6 OPM Process Performance Controlling model

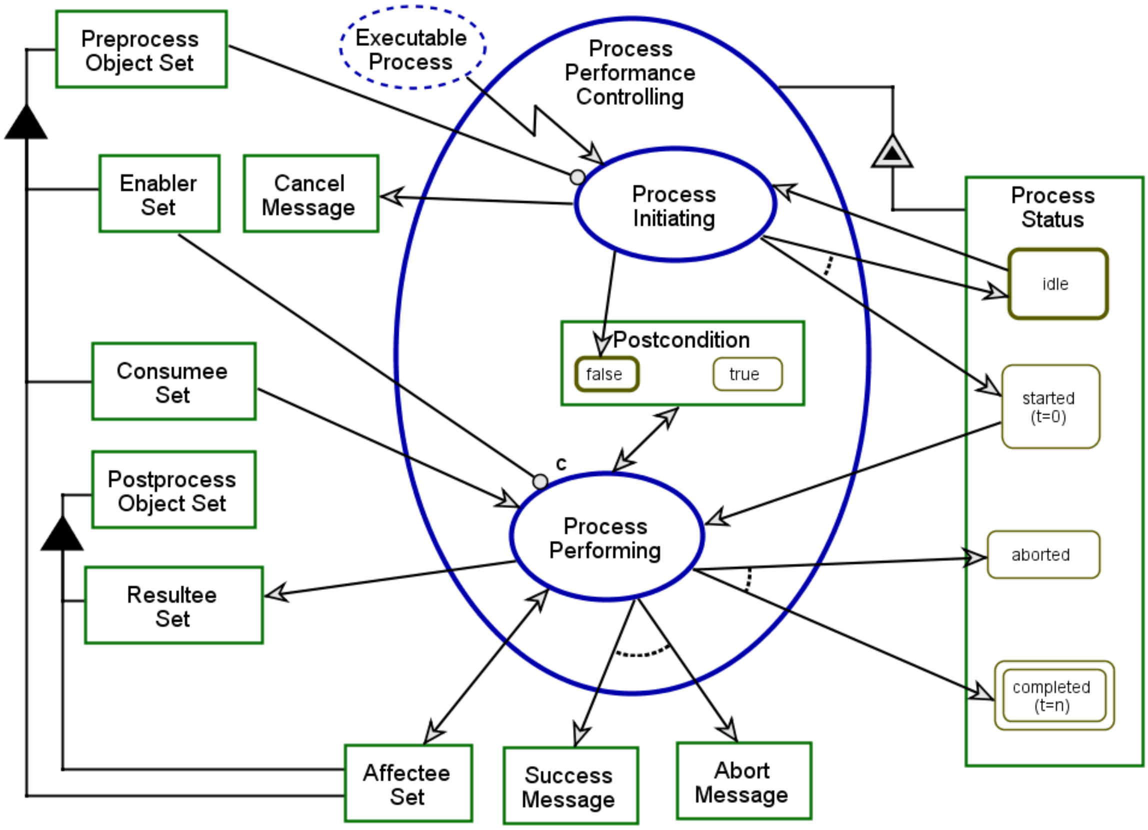
C.6.1 OPM Process Performance Controlling System - SD



Involved Object Set consists of **Preprocess Object Set** and **Postprocess Object Set.**
Preprocess Object Set exhibits **Size.**
Size of **Preprocess Object Set** is **r>=0.**
Postprocess Object Set exhibits **Size.**
Size of **Postprocess Object Set** is **s>=0.**
Involved Object Set exhibits **Size.**
Size of **Involved Object Set** is **r+s>=0.**
Process Performance Controlling affects **Involved Object Set.**
Executable Process is **environmental.**
Executable Process invokes **Process Performance Controlling.**
Process Performance Controlling yields one of **Success Message** or **Failure Message.**
Abort Message and **Cancel Message** are **Failure Messages.**

Figure C.22 — Process Performance Controlling system diagram – SD

C.6.2 Process Performance Controlling in-zoomed as SD1



Process Performance Controlling zooms into **Process Initiating** and **Process Performing** in that sequence, as well as **Postcondition.**

Preprocess Object Set consists of **Consumee Set, Affectee Set**, and **Enabler Set.**

Postprocess Object Set consists of **Resultee Set** and **Affectee Set.**

Executable Process is environmental.

Executable Process invokes **Process Initiating.**

Process Performance Controlling exhibits **Process Status.**

Process Status can be **idle, started (t=0), aborted**, or **completed (t=n).**

Process Status is initially **idle** and finally **completed (t=n)** or **aborted.**

Postcondition can be **false** or **true.**

Postcondition is initially **false.**

Process Initiating requires **Preprocess Object Set.**

Process Initiating changes **Process Status** from **idle** to one of **idle** or **started (t=0).**

Process Initiating yields **false Postcondition** and **Cancel Message.**

Process Performing occurs if **Enabler Set** exists, otherwise **Process Performing** is skipped**.**

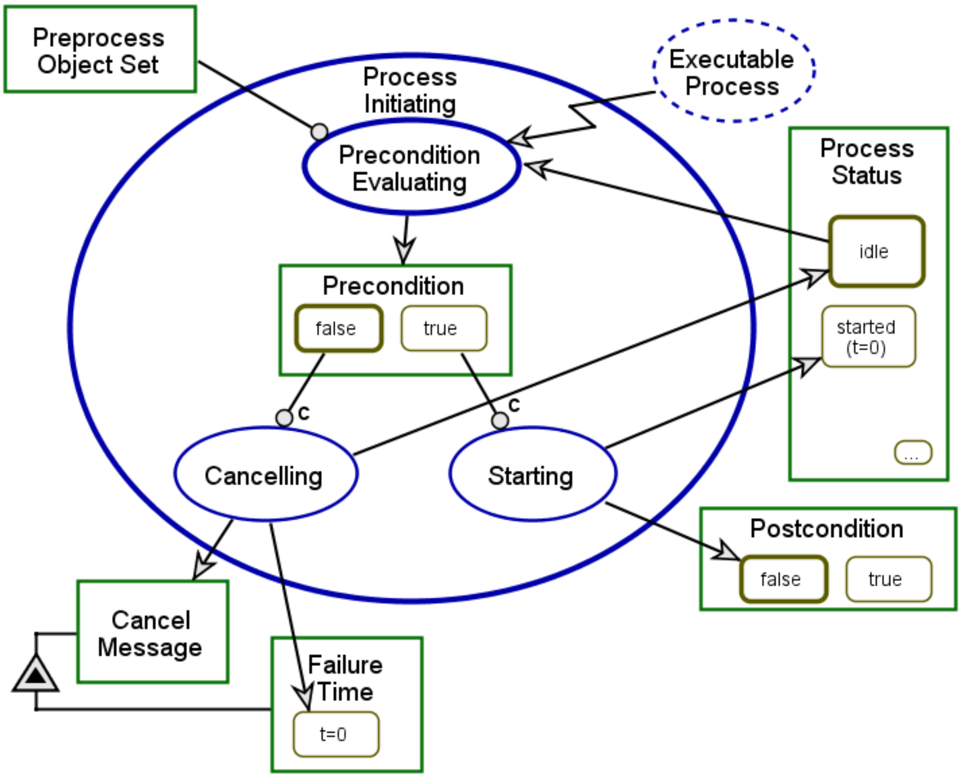
Process Performing affects **Postcondition** and **Affectee Set.**

Process Performing changes **Process Status** from **started (t=0)** to one of **aborted** or **completed (t=n).**

Process Performing yields **Resultee Set** and either **Success Message** or **Abortion Message.**

Figure C.23 — Process Performance Controlling from SD in-zoomed in SD1

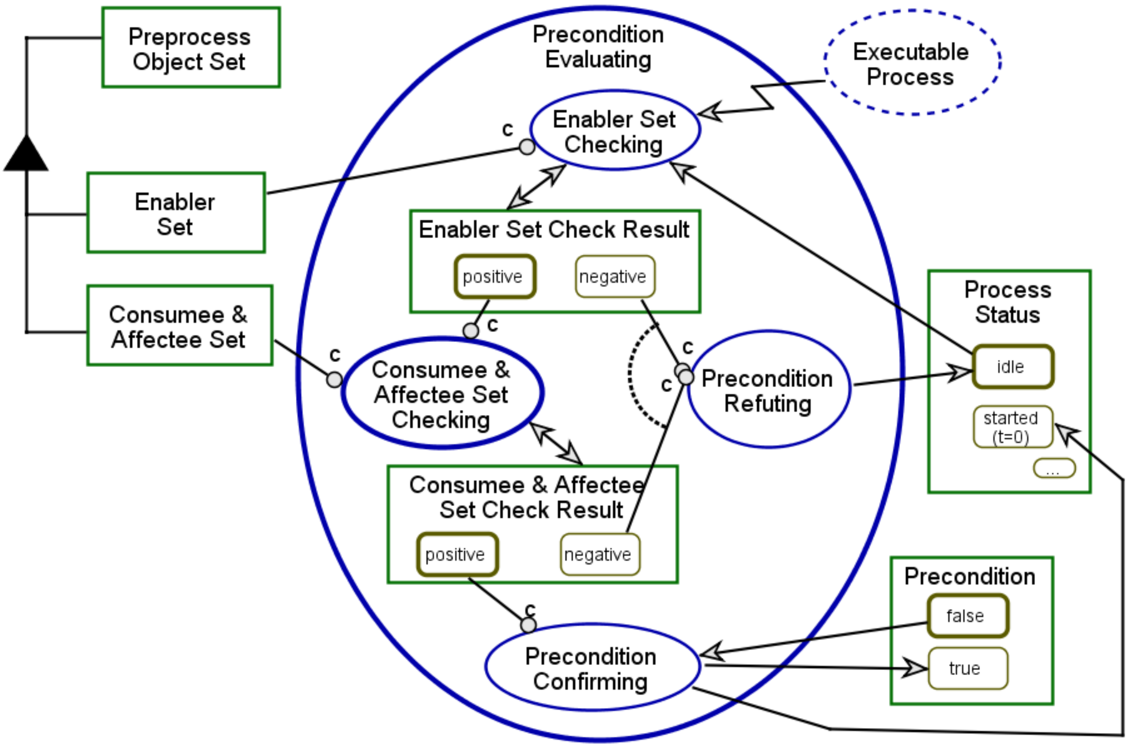
C.6.3 Process Initiating in-zoomed as SD1.1



Process Initiating from **SD1** zooms in **SD1.1** into **Precondition Evaluating** and **parallel Cancelling** and **Starting,** in that sequence, as well as **Precondition.**
Process Status can be **idle, started (t=0)**, or other **states.**
Process Status is initially **idle.**
Postcondition can be **false** or **true.**
Postcondition is initially **false.**
Executable Process is environmental.
Executable Process invokes **Precondition Evaluating.**
Precondition Evaluating yields **Precondition.**
Precondition can be **true** or **false.**
Precondition Evaluating requires **Preprocess Object Set.**
Precondition Evaluating changes **Process Status** from **idle.**
Cancelling occurs if **Precondition** is **false,** otherwise **Cancelling** is skipped**.**
Cancelling changes **Process Status** to **idle.**
Cancelling yields **Cancel Message.**
Cancellation Message exhibits **Failure time.**
Cancelling sets the value of **Failure time** to **t=0.**
Failure time of **Cancel Message** is **t=0.**
Starting occurs if **Precondition** is **true**, in which case **Precondition** is consumed, otherwise **Starting** is skipped.
Starting changes **Process Status** to **started (t=0).**
Starting yields **false Postcondition.**

Figure C.24 — Process Initiating in-zoomed as SD1.1

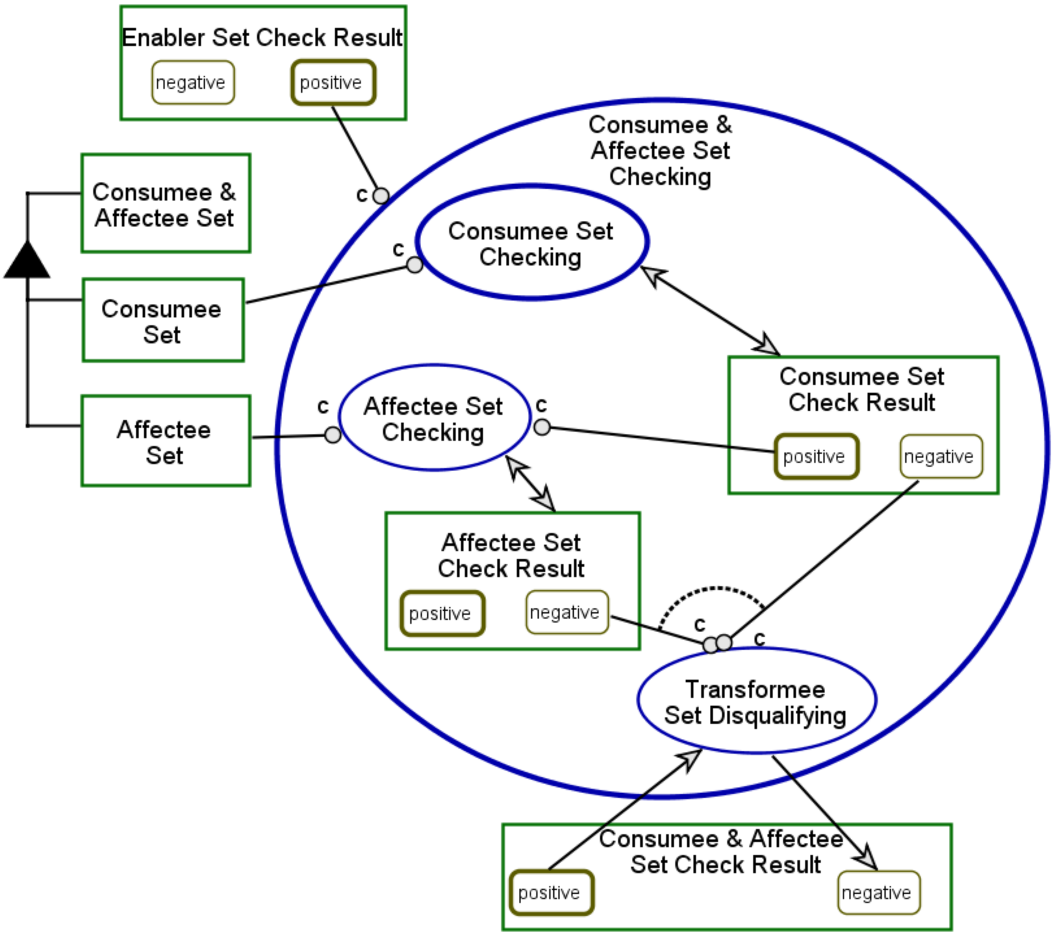
C.6.4 Precondition Evaluating in-zoomed as SD1.1.1



Precondition Evaluating from **SD1.1** zooms in **SD1.1.1** into **Enabler Set Checking,** **Consumee & Affectee Set Checking, Precondition Refuting**, and **Precondition Confirming** in that sequence, as well as **Enabler Set Check Result** and **Consumee & Affectee Set Check Result.**
Preprocess Object Set consists of **Enabler Set** and **Consumee & Affectee Set.**
Process Status can be **idle, started (t=0)**, or **other states.**
Process Status is initially **idle.**
Precondition can be **false** or **true.**
Precondition is initially **false.**
Executable Process invokes **Enabler Set Checking.**
Enabler Set Checking requires that **Enabler Set** exists, otherwise **Enabler Set Checking** is skipped**.**
Enabler Set Checking changes **Process Status** from **idle.**
Enabler Set Check Result can be **positive** or **negative.**
Enabler Set Check Result is initially **positive.**
Enabler Set Checking affects **Enabler Set Check Result.**
Consumee & Affectee Set Checking occurs if **Enabler Set Check Result** is **positive** and **Consumee & Affectee Set** exists, otherwise **Consumee & Affectee Set Checking** is skipped**.**
Consumee & Affectee Set Check Result can be **positive** or **negative.**
Consumee & Affectee Set Check Result is initially **positive.**
Consumee & Affectee Set Checking affects **Consumee & Affectee Set Check Result.**
Precondition Refuting requires that either **Enabler Set Check Result** is **negative** or **Consumee & Affectee Check Result** is **negative,** otherwise **Precondition Refuting** is skipped**.**
Precondition Refuting changes **Process Status** to **idle.**
Precondition Confirming occurs if **Transformee Check Result** is **positive,** otherwise **Precondition Confirming** is skipped**.**
Precondition Confirming changes **Precondition** from **false** to **true** and **Process Status** to **started (t=0).**

Figure C.25 — Precondition Evaluating in-zoomed – SD1.1.1

C.6.5 Transformee Set Checking in-zoomed as SD1.1.1.1



Consumee & Affectee Set Checking from **SD1.1.1** zooms in **SD1.1.1** into **Consumee Set Checking, Affectee Set Checking**, and **Transformee Set Disqualifying** in that sequence, as well as **Affectee Set Check Results** and **Consumee Set Check Results.**

Enabler Set Check Result can be **negative** or **positive.**

Enabler Set Check Result is initially **positive.**

Consumee & Affectee Set Check Result can be **negative** or **positive.**

Consumee & Affectee Set Check Result is initially **positive.**

Consumee & Affectee Set consists of **Consumee Set** and **Affectee Set.**

Consumee & Affectee Set Checking occurs if **Enabler Set Check Result** is **positive**, otherwise **Consumee & Affectee Set Checking** is skipped**.**

Consumee Set Check Results can be **negative** or **positive.**

Consumee Set Check Results is initially **positive.**

Consumee Set Checking occurs if **Consumee Set** exists, otherwise **Consumee Set Checking** is skipped**.**

Consumee Set Checking affects **Consumee Set Check Results.**

Affectee Set Checking occurs if **Consumee Set Consumee Set Check Results** is **positive and Affectee Set** exists, otherwise **Affectee Set Checking** is skipped**.**

Affectee Set Checking yields **Affectee Set Check Results.**

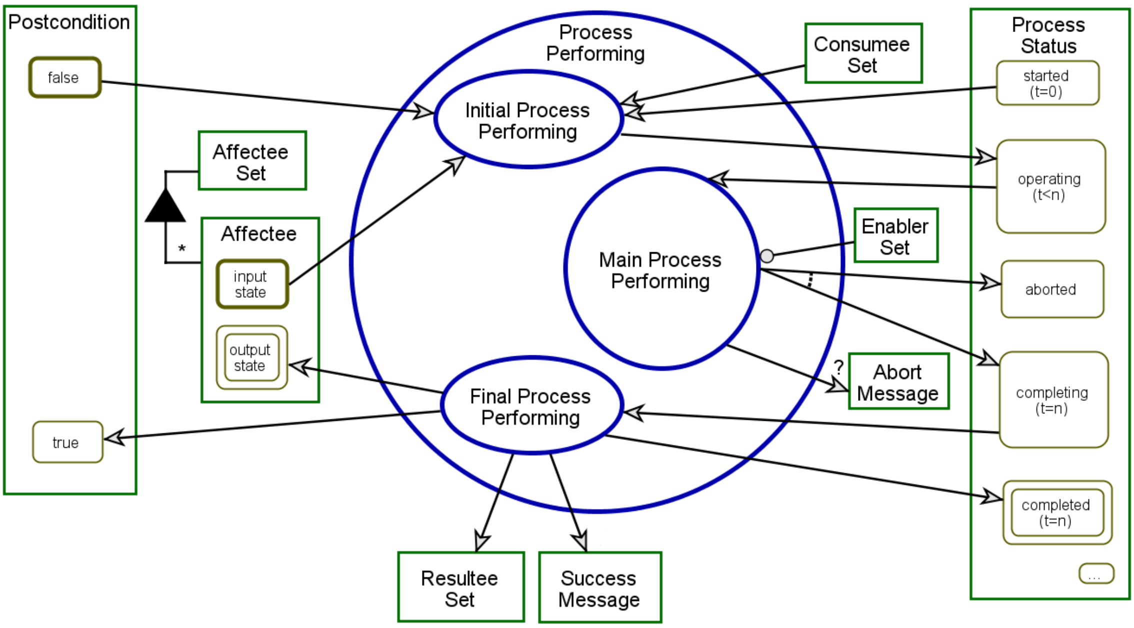
Affectee Set Check Results can be **negative** or **positive.**

Transformee Set Disqualifying occurs if either **Affectee Set Check Results** is **negative** or **Consumee Set Check Results** is **negative.**

Transformee Set Disqualifying changes **Consumee & Affectee Set Check Result** from **positive** to **negative.**

Figure C.26 — Transformee Set Checking in-zoomed – SD1.1.1.1

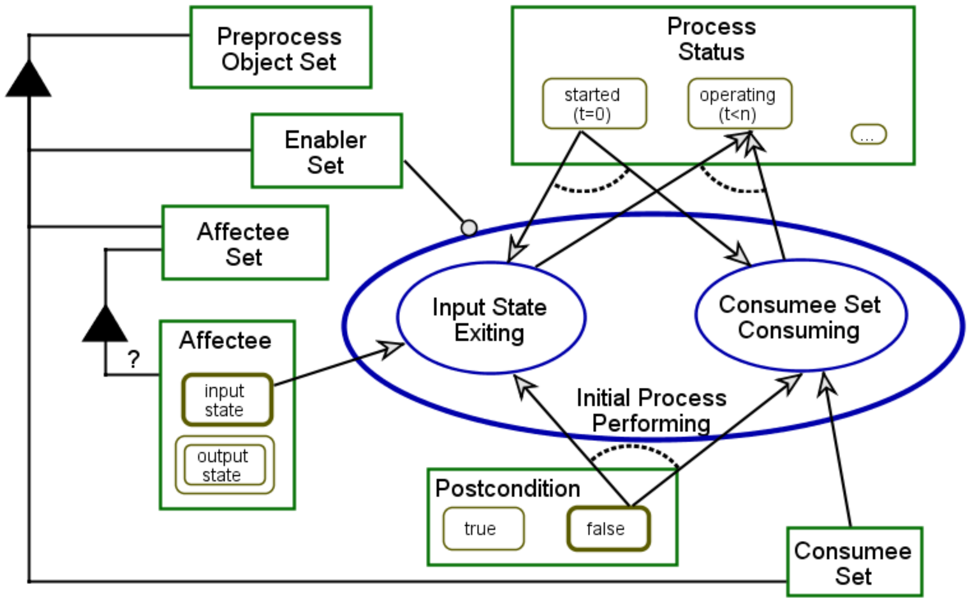
C.6.6 Process Performing in-zoomed as SD1.2



Process Performing from **SD1** zooms in **SD1.2** into **Initial Process Performing, Main Process Performing**, and **Final Process Performing** in that sequence**.**
Process Status can be **idle, started (t=0), operating (t<n), aborted, completing (t=n), completed (t=n)**, or **other states.**
Process Status is finally **completed (t=n).**
Postcondition can be **false** or **true.**
Postcondition is initially **false.**
Affectee Set consists of **optional Affectees.**
Affectee can be **input state** or **output state.**
Affectee is initially **input state** and finally **output state.**
Initial Process Performing changes **Process Status** from **started (t=0)** to **operating (t<n), Postcondition** from **false,** and **Affectee** from **input state.**
Initial Process Performing consumes **Consumee Set.**
Main Process Performing requires **Enabler Set.**
Main Process Performing yields an optional **Abort Message.**
Main Process Performing changes **Process Status** from **operating (t<n)** to one of **completing (t=n)** or **aborted.**
Final Process Performing changes **Process Status** from **completing (t=n)** to **completed (t=n), Postcondition** to **true,** and **Affectee** to **output state.**
Final Process Performing yields **Success Message** and **Resultee Set.**

Figure C.27 — Process Performing in-zoomed – SD1.2

C.6.7 Initial Process Performing in-zoomed as SD1.2.1



Initial Process Performing from **SD1.2** zooms in **SD1.2.1** into parallel **Input State Exiting** and **Consumee Set Consuming.**

Preprocess Object Set consists of **Enabler Set, Affectee Set**, and **Consumee Set.**

Affectee Set consists of optional **Affectees.**

Affectee can be **input state** or **output state.**

Affectee is initially **input state** and finally **output state.**

Process Status can be **started (t=0), operating (t<0)**, or other states**.**

Postcondition can be **false** or **true.**

Postcondition is initially **false.**

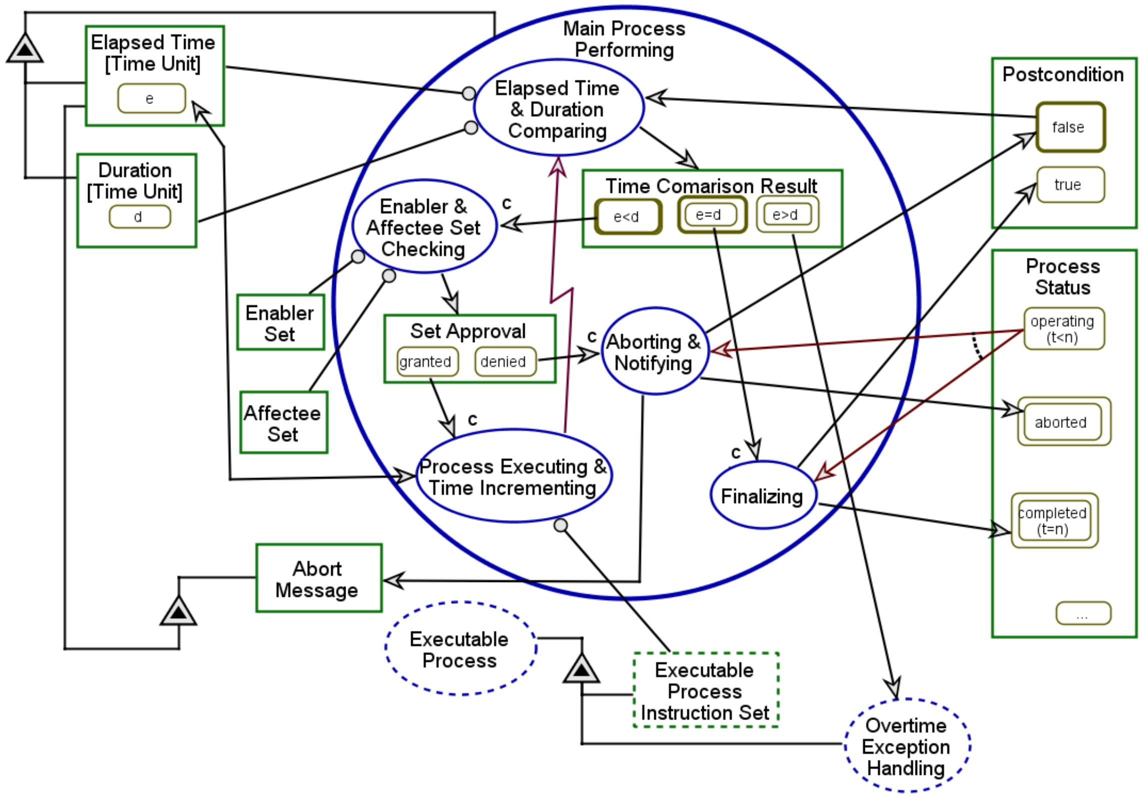
Initial Process Performing requires **Enabler Set.**

Input State Exiting changes **Affectee** from **input state.**

One of **Consumee Set Consuming** or **Input State Exiting** changes **Process Status** from **started (t=0)** to **operating (t<n)** and **Postcondition** from **false.**

Figure C.28 — Initial Process Performing in-zoomed – SD1.2.1

C.6.8 Main Process Performing in-zoomed as SD1.2.2



Main Process Performing from **SD1.2** zooms in **SD1.2.2** into **Elapsed Time & Duration Comparing, Enabler & Affectee Set Checking, Aborting & Notifying, Time Incrementing**, and **Finalizing**, in that sequence, as well as **Time Comparison Result** and **Set Approval.**

Executable Process exhibits **Executable Process Instruction Set** and **Overtime Exception Handling.**

Executable Process, Executable Process Instruction Set, and **Overtime Exception Handling** are **environmental.**

Process Status can be **aborted, completed (t=n), operating (t<0)** or other states**.**

Process Status is finally **aborted** or **completed (t=n).**

Postcondition can be **false** or **true.**

Postcondition is initially **false.**

Main Process Performing exhibits **Elapsed Time** in **Time Unit** and **Duration** in **Time Unit.**

Abortion Message exhibits **Elapsed Time** in **Time Unit.**

Elapsed Time in **Time Unit** is **e.**

Duration in **Time Unit** is **d.**

Elapsed Time & Duration Comparing requires **Elapsed Time** in **Time Unit** and **Duration** in **Time Unit.**

Elapsed Time & Duration Comparing changes **Postcondition** from **false.**

Elapsed Time & Duration Comparing yields **Time Comparison Result.**

Time Comparison Result can be **e<d, e=d**, or **e>d.**

Time Comparison Result is initially **e<d** or **e=d** and **finally e=d** or **e>d.**

Enabler & Affectee Set Checking requires **Enabler Set** and **Affectee Set.**

Enabler & Affectee Set Checking occurs if **Time Comparison Result** is **e<d**, in which case **Enabler & Affectee Set Checking** consumes **Time Comparison Result**, otherwise **Enabler & Affectee Set Checking** is skipped**.**

Enabler & Affectee Set Checking requires **Enabler Set.**

Enabler & Affectee Set Checking yields **Set Approval.**

Set Approval can be **granted** or **denied.**

Aborting & Notifying occurs if **Set Approval** is **denied,** in which case **Aborting & Notifying** consumes **Set Approval**, otherwise **Aborting & Notifying** is skipped**.**

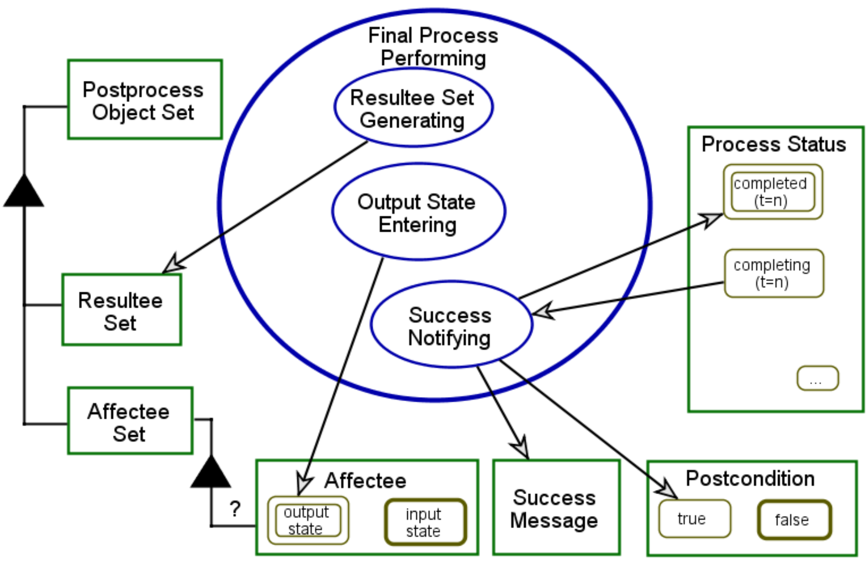
Aborting & Notifying changes Process Status from **operating (t<n)** to **aborted** and **Postcondition** to **false.**

Aborting & Notifying yields **Abort Message.**

Abort Message Finalizing occurs if **Time Comparison Result** is **e=d**, in which case **Finalizing** consumes **Time Comparison Result**, otherwise **Finalizing** is skipped**.**
Finalizing changes **Process Status** from **operating (t<n)** to **completed (t=n)** and **Postcondition** to **true.**
Process Executing & Time Incrementing requires **Executable Process Instruction Set.**
Process Executing & Time Incrementing occurs if **Set Approval** is **granted,** in which case **Process Executing & Time Incrementing** consumes **Set Approval**, otherwise **Process Executing & Time Incrementing** is skipped**.**
Time Incrementing consumes **Sets are OK?.**
Time Incrementing yields **elt=1..ext Elapsed Time** in **Time Unit.**
Process Executing & Time Incrementing changes **the value e of Elapsed Time** in **Time Unit.**
Process Executing & Time Incrementing invokes **Elapsed Time & Duration Comparing.**
Overtime Exception Handling consumes **e>d Time Comparison Result.**

Figure C.29 — Main Process Performing in-zoomed – SD1.2.2

C.6.9 Final Process Performing in-zoomed as SD1.2.3



Final Process Performing from **SD1.2** zooms in **SD1.2**.3 into parallel **Resultee Set Generating, Output State Entering**, and **Success Notifying**, in that sequence**.**
Postprocess Object Set consists of **Resultee Set** and **Affectee Set.**
Affectee Set consists of optional **Affectees.**
Affectee can be **input state** or **output state.**
Affectee is initially **input state** and finally **output state.**
Process Status can be **completed (t=n), completing (t=n)**, or other states**.**
Process Status is finally **completed (t=n).**
Postcondition can be **false** or **true.**
Postcondition is initially **false.**
Resultee Set Generating yields **Resultee Set.**
Output State Entering changes **Affectee** to **output state.**
Success Notifying changes **Postcondition** to **true.**
Success Notifying yields **Success Message.**

Figure C.30 — Final Process Performing in-zoomed – SD1.2.3

Annex D
(informative)

OPM dynamics and simulation

D.1 OPM executability

An OPM model provides for executability—the ability to simulate a system by executing its model via animation in a properly designed software environment.

D.2 Change and effect

A change of an object is an alteration in the state of that object. More specifically, a change of an object is reflected by replacing its current state by another state. The only thing that can cause this change is a process. The process causes the change by taking as input an object at some state, and outputting it in another state. Hence, a change of an object means a change in the state at which the object is at.

Stateful objects can be affected, i.e. their states can change. This change mechanism underlines the intimate, inseparable link between objects and processes. This change in state is the effect of the process on the object.

Effect is therefore defined as the change in the state of an object that a process causes.

While the terms "change" and "effect" are almost synonymous, there is a subtle difference in their usage. Effect Is used to refer to what the process does to the object, and change—to what happens to the object as a result of the process occurrence. Later in this section the above definition of effect is refined with the notions of input and output links.

D.3 Existence and transformation

Change is only one possibility of what can happen to an object when a process acts on it. A process affects an object to change it, but it can also do things that are more drastic: it can generate an object or consume it. The term transformation covers these three additional modes by which a process can act on an object: construction, effect, and consumption.

Construction is synonymous with creation, generation, or yielding. Effect is synonymous with change or switch, and consumption is synonymous with elimination, termination, annihilation, or destruction. The effect of a process on an object is to change that object from one of its states to another, but the object still exists, and it keeps maintaining the identity it had before the process occurred. Construction and consumption change the very existence of the object and are therefore more profound transformations than effect.

When a process constructs (yields, generates, creates, or results in) an object, the meaning is that the object, which had not previously existed, has undergone a radical transformation. This transformation made it stand out and become identifiable and meaningful in the system. It now deserves treatment and reference as a new, separate entity.

When a process consumes (eliminates or destroys) an object, the meaning is that the object, which had previously existed, and was identifiable and meaningful in the system, has undergone a radical transformation. Consequently, the object no longer exists in the system and is no longer identifiable.

D.4 Timeline OPM principle

By default the execution timeline within an in-zoomed process begins at the graphical top and ends at the graphical bottom unless there is indication to deviate from the timeline. Such indications include the special OPM process **Exiting**, discussed below, and internal events within the scope of the process that can cause loops.

The top-most point of the process ellipse serves as a reference point, so a process whose reference point is higher that its peer(s) starts earlier. If the reference points of two or more processes are at the same height (within a few graphical units, e.g pixels, of tolerance), these processes start simultaneously and in parallel.

D.5 Timed events

The events presented so far were object or state events: they happened when a specific object became existent or entered a specific state. In contrast, timed events depend on the arrival of a specific time in the system, as shown below.

A state event can represent a time event, as Figure D.1 — Legal system model change from minor to adult at the Age of 18 Years demonstrates.

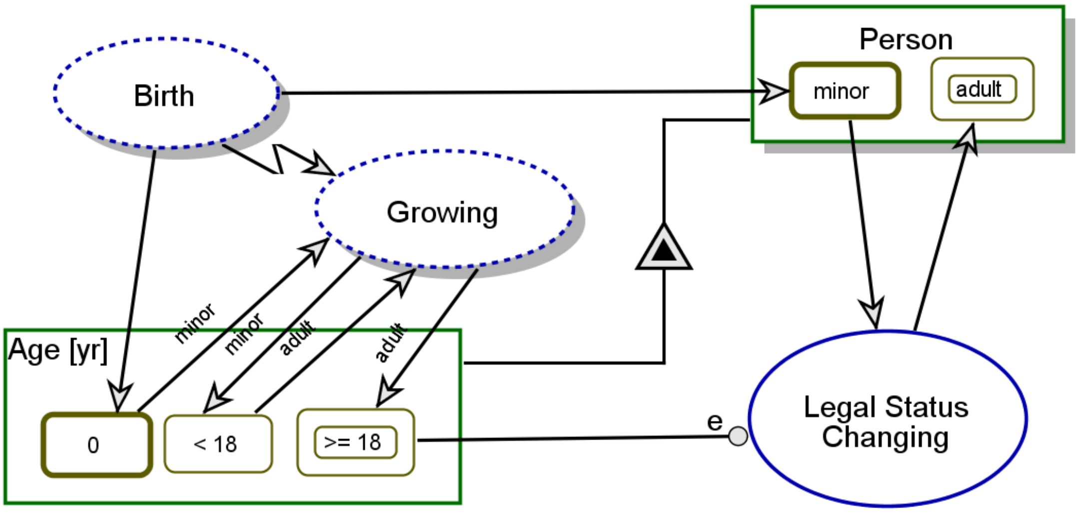


Figure D.1 — Legal system model change from minor to adult at the Age of 18 Years

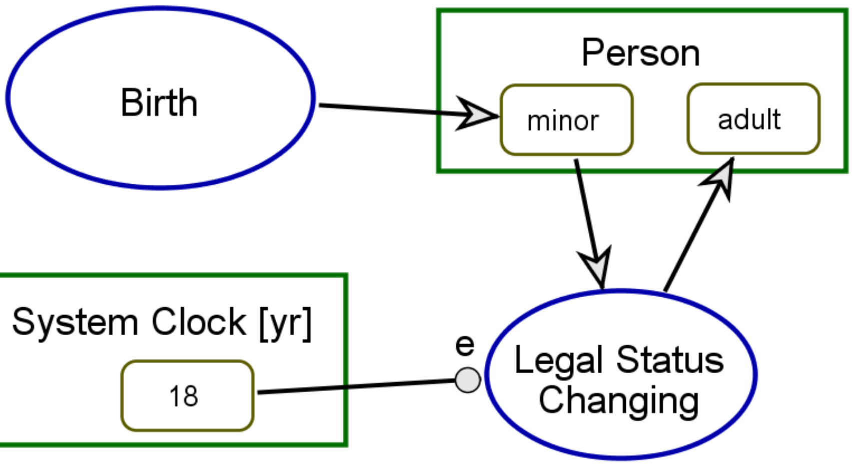


Figure D.2 — The System Clock event initiating Legal Status Changing

D.6 Object history and the lifespan diagram

At any point in time, an object can be in one of its states, or exists in transition between two states.

A lifespan diagram is a diagram showing for any point in time during the life of the system what objects exists in the system, what state each object is at, and what processes are active.

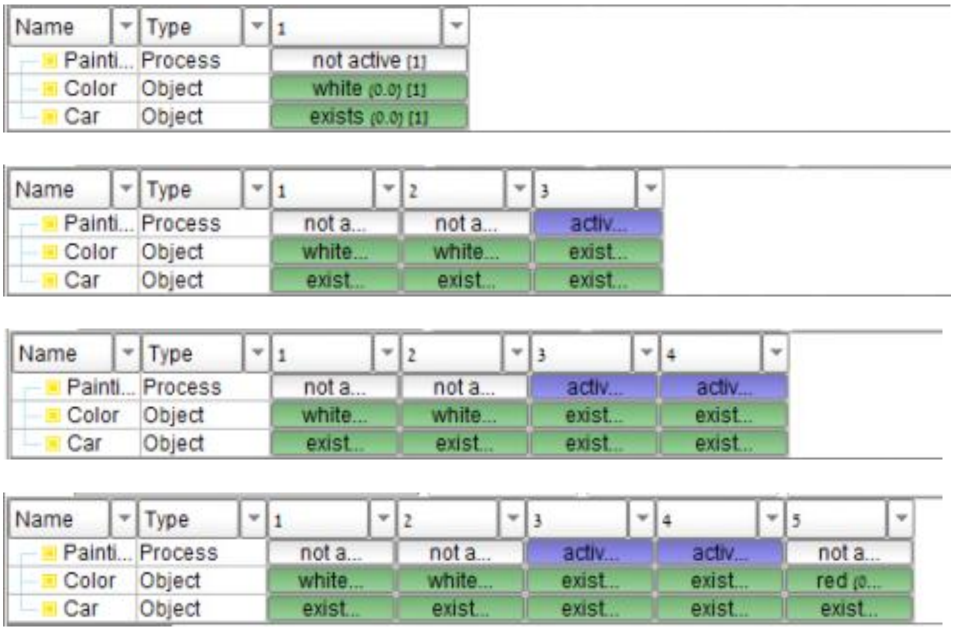


Figure D.3 — Car Painting four lifespan diagrams example

The four lifespan diagrams shown at Figure D.3 — Car Painting four lifespan diagrams example record the history of the car painting system as time progresses. These four lifespan diagrams are displayed stacked vertically to facilitate their inspection. In the first diagram, only the first time period is displayed. Painting is not active, and the Car is white.

In the second diagram, the first three time periods are displayed. In the third period, Painting is active, and the Car is no longer white. The same happens in the fourth period, as shown in the third diagram. Finally, in the fifth period, shown in the bottom diagram, Painting is no longer active, and the Car is red.

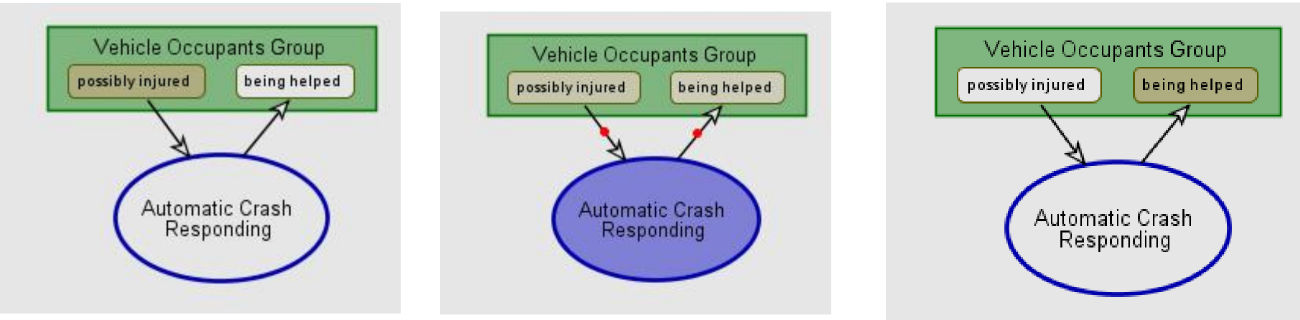


Figure D.4 — Executing the OPM model for Automatic Crash Responding

Figure D.4 — Executing the OPM model for Automatic Crash Responding presents three OPCAT screenshots, showing three stages of executing an OPM model. The screenshot on the left hand side shows the system before the **Automatic Crash Responding** process occurs. At this stage, **Vehicle Occupants Group** is at its input state, **possibly injured**, and this is marked by the state being solid (coloured brown).

The middle screenshot shows the process in action, marked as solid (coloured blue). During the time that the process **Automatic Crash Responding** is active (i.e. when it executes), the object **Vehicle Occupants Group** is in transition from its input state, **possibly injured**, to its output state, **being helped**. This is marked by both states being semi-solid.

Observing the animation in action, the input state is gradually fading out while the output state is becoming solid. At the same time, two red dots travel along the input-output link pair, denoting the "control" of the system, or where the system is at each time point. One red dot travels from the input state to the affecting process. At the same time, the second dot travels from that process along the output link to the output state.

Finally, the screenshot on the right shows the system after the **Automatic Crash Responding** process had terminated. At this stage, **Vehicle Occupants Group** is at its output state, **being helped**.

The animated execution of the system model has several benefits. First, it is a dynamic visualization aid that helps both the modeller and the target audience follow and understand the behaviour of the system over time. Second, like a debugger of a programming language, it facilitates verification of the system's dynamics and spotting logical design errors in its flow of execution control. Therefore, frequently animating the system model during its construction is highly recommended.

D.7 Process duration

System time unit is the default time unit used for specifying all duration kinds of all the processes in the system unless there is an explicit different time unit for a specific process, in which case that time unit overrides the system time unit.

A compact way to express the relevant process property values in an OPD uses exhibition-characterization and specialization links. Assuming that the following are relevant process properties, EXAMPLE 1 expresses two ways to graphically configure the properties:

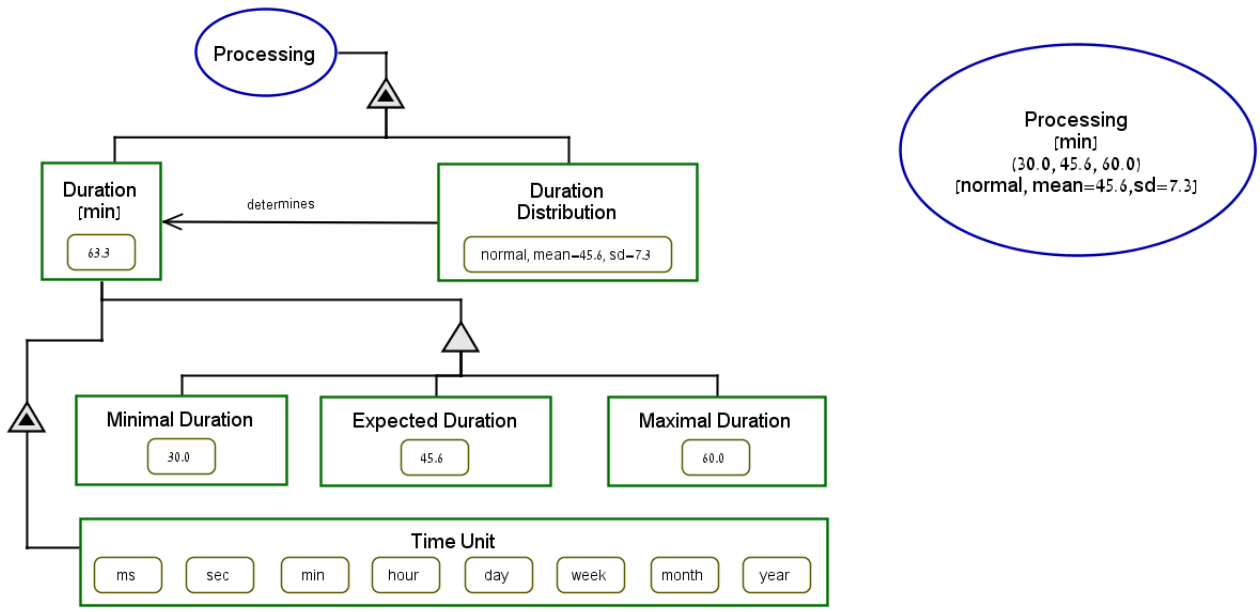
- the time measurement unit;
- time duration parameters, which can be one of the following:
  - three values, standing for the minimal, expected, and maximal duration, respectively,
  - two values, standing for the minimal and maximal duration, respectively, or
  - one value, standing for both the minimal and maximal durations; and,
- the duration distribution name and its one or more parameters.

The following are possible normative distributions and their parameter(s):

- Normal, mean=xx; sd=yy;
- Uniform, a=xx, b=yy; and,
- Exponential, lambda=xx.

NOTE The time measurement unit of seconds, abbreviated as sec, is the customary default and often omitted.

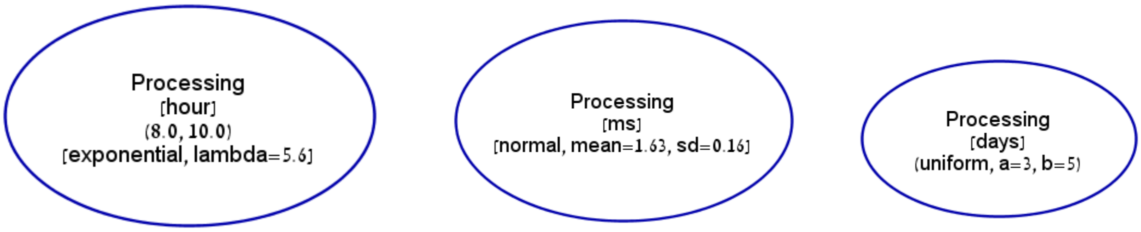
EXAMPLE 1 is a metamodel of Processing Duration with property values. On the left is the complete metamodel. The process on the right shows a compact way to record all the data on the left, except for the (actual) Duration, which is a run-time property. The Duration Distribution in this example is normal with mean 45.6 minutes and standard deviation 7.3 minutes.



Processing exhibits **30.0**, **45.6**, and **60.0 min Minimal Duration**, **Expected Duration**, and **Maximal Duration**, respectively and **normal Duration Distribution** with parameters **mean=45.6** and **sd=70.0**.

Figure D.5 — Processing Duration with property values

EXAMPLE 2



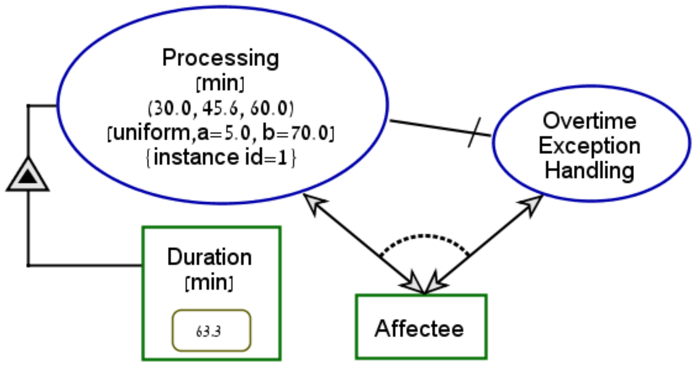
Processing exhibits **8.0** and **10.0 hour Minimal Duration** and **Maximal Duration**, respectively, and **exponential Duration Distribution** with parameter **lambda=5.6.**

Processing exhibits **normal Duration Distribution** with parameters **mean=1.63** and **sd=0.16 ms.**

Processing exhibits **uniform Duration Distribution** with parameters **a=3** and **b=5 days.**

Figure D.6 — Process duration examples

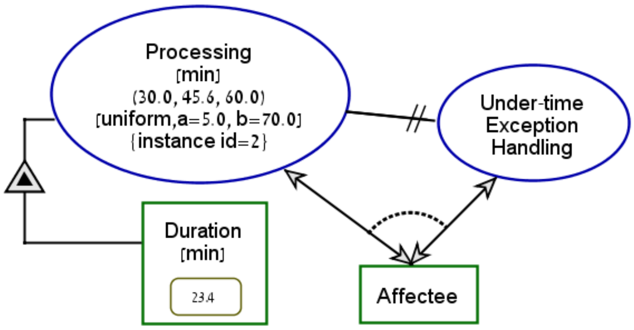
EXAMPLE 3 In Figure D.7 — Overtime exception example, Processing {instance id=1} Duration is 63.3 min, hence Overtime Exception Handling occurs.



Processing exhibits **30.0**, **45.6,** and **60.0 min Minimal Duration, Expected Duration,**
and **Maximal Duration**, respectively, and **uniform Duration Distribution** with parameters **a=5.0 and b=70.0.**
Either **Processing** or **Overtime Exception Handling** affects **Affectee.**
Overtime Exception Handling occurs if duration of **Processing** exceeds **60.0 min.**
Overtime Exception Handling affects Affectee.

Figure D.7 — Overtime exception example

EXAMPLE 4 In Figure D.8 — Undertime exception example, Processing {instance id=2} Duration is 23.4 min, hence Undertime Exception Handling occurs.



Processing exhibits **30.0**, **45.6,** and **60.0 min Minimal Duration, Expected Duration,**
and **Maximal Duration**, respectively, and **uniform Duration Distribution** with parameters **a=5.0 and b=70.0.**
Either **Processing** or **Undertime Exception Handling** affects **Affectee.**
Undertime Exception Handling occurs if duration of **Processing** falls short of **60.0 min.**
Undertime Exception Handling affects Affectee.

Figure D.8 — Undertime exception example

Annex E
(informative)

Graph grammar of OPM

E.1 Graph grammar overview

An OPD graph is a bipartite graph with two node kinds, objects and processes, connected by various kinds of edge, i.e., links. Annex F describes a graph grammar for the creation of valid diagrams in the Object-Process Methodology visual modelling notation (Object-Process Diagrams).

Graph Grammars (or Graph Transformations) is a field of Graph Theory that formalizes the creation or transformation of graphs using predefined transformation rules. Informally, a graph grammar consists of a set of productions that, when applied to a diagram, add to or modify the diagram. A production consists of a source and target graphs and a morphism that defines the transformation from the source graph to the target graph. Figure E.1 — Example of graph production shows an example of a production.

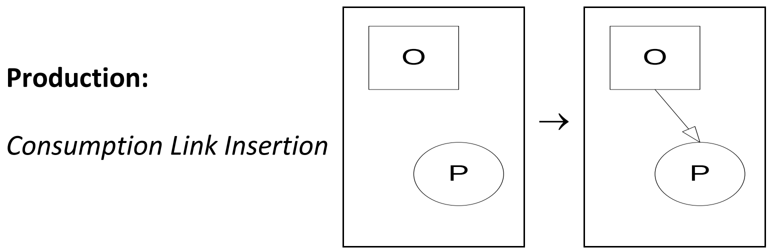


Figure E.1 — Example of graph production

The production shown in the example describes a production to create a consumption link between an object and a process. Figure E.2 — Base diagram for use of a production show a base OPD diagram for application of the product, sometimes referred to as a derivation, from Figure E.1 — Example of graph production.

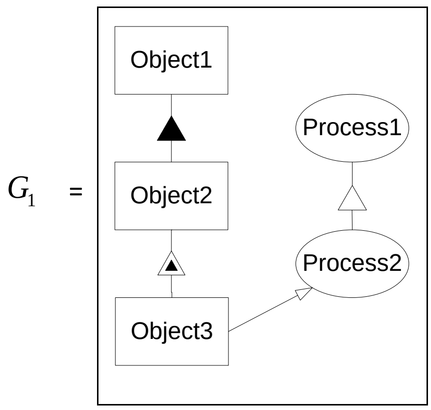


Figure E.2 — Base diagram for use of a production

To apply the production, one matches the elements in the source graph of the production with elements in the existing OPD. Following OPD conventions, **O** matches to **Object1**, **Object2**, and **Object3**. **P** matches in a similar fashion. After selecting a match (many matches can be found, therefore one is chosen), the production is applied to the OPD. Suppose selection of the pair **Object1**, **Process1** occurs, then the derivation changes the OPD by adding it a new consumption link as shown in Figure E.3 — Applying a production to a diagram.

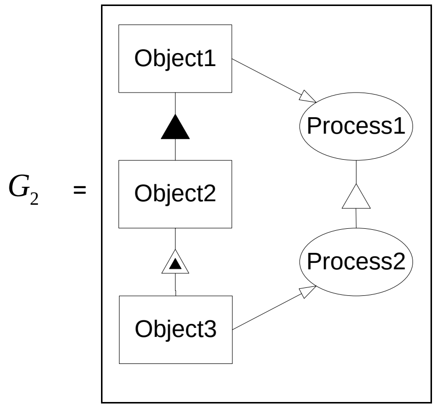


Figure E.3 — Applying a production to a diagram

Productions may be conditional, so that their application is constrained by the current context of application. Given that OPM does not allow for two consumption links between an object and a process, the graph grammar defines a conditional production as show in Figure E.4 — Graph grammar constraint for consumption link. This production defines that a new consumption link can occur between an object and a process, but only when this link does not exist already (as shown by the shadowed link in the source of the production).

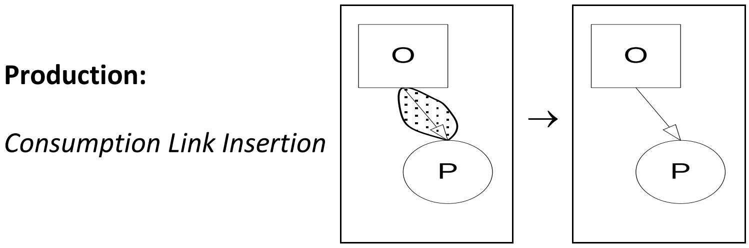


Figure E.4 — Graph grammar constraint for consumption link

A partial graph grammar for the creation of OPDs is defined in [4] and a short description of the grammar defined there will be shown below.

NOTE The reader interested in the complete definition is invited to read the original source. Also, more information on Graph Grammars can be found in (Corradini, A.; Ehrig, H.; Heckel, R.; Korff, M.; Lowe, M.; Ribeiro, L. & Wagner, A. (1997), Algebraic Approaches to Graph Transformation - Part I: Single Pushout Approach and Comparison with Double Pushout Approach, in G. Rozenberg, ed.,'Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations', World Scientific, pp. 247-312) and (Ehrig, H.; Heckel, R.; Korff, M.; Lцwe, M.; Ribeiro, L.; Wagner, A. & Corradini, A. (1997), Algebraic approaches to graph transformation. Part II: single pushout approach and comparison with double pushout approach, in 'Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations', World Scientific, pp. 247-312.)

E.2 Using graph grammars in OPD

E.2.1 Proactive and reactive stages

The creation of an OPD using graph grammars occurs in two stages: proactive and reactive. In the proactive stage the user creates a diagram following the graph grammar rules outlined in this Annex. The proactive creation process allows for temporary inconsistencies in the OPD, which enable easy modelling while maintaining a general consistency in the diagram. After creating a model, the modeller can apply the reactive stage, which validates that the existing OPD is completely valid. Because the reactive stage is applicable anytime during the modelling process, the determination of the validity of every change to the diagram is possible.

This Annex presents a number of preliminary definitions useful in both the proactive and reactive stages of OPD creation, then identifies the proactive stage as OPD Creation, and finally describes the reactive stage as OPD Validation.

E.2.2 Preliminary definitions

E.2.2.1 Abstract link

An abstract link is an OPM link that stands for any type of concrete link that can connect two element in the model. Its graphical representation is a straight line drawn between the two elements, as shown Figure E.5 — Abstract link between two things, and a state and a thing.

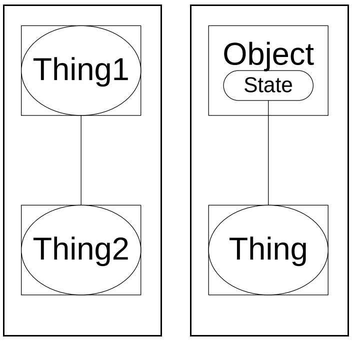


Figure E.5 — Abstract link between two things, and a state and a thing

An abstract link is undirected, unless an open arrow appears ends. Since in OPM this is the symbol for the tagged structural relation, the tagged structural relations symbol changes to a double arrowhead by using the relevant rule to remove the ambiguity.

For convenience, an abstract link specializes into an abstract structural or procedural link by adding the letter "s" or "p" to the link.

E.2.2.2 Modelling conventions

The remainder of this Annex uses the following notational conventions:

- A negative constraint appears as shaded areas in the appropriate context within the left-hand graph of the production.
- Elements in the rules are named as follows:
  - *Thing*: T (if only one appearance exists in the OPD), T1, T2 ...
  - *Object*: O (only one appearance), O1, O2 ...
  - *Process*: P (only one appearance), P1, P2 ...
  - *States*: s (only one appearance), s1, s2 ...

E.2.3 OPD creation – productions

This section shows the 13 primary productions for use to build an OPD from scratch.

1) *Thing* creation: add new *things* to the OPD for two situations –

   i) If there is no *thing* with the same name as the *thing* added.

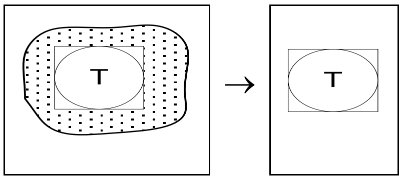


Figure E.6 — Creating a new thing production

ii) If there is a *thing* with the same name but the existing *thing* has a structural parent defined in the OPD.

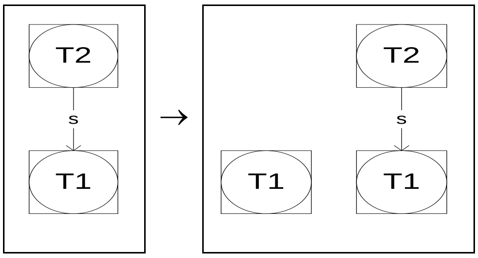


Figure E.7 — Creating the same thing twice production

2) State creation: add a *state* to an existing object.

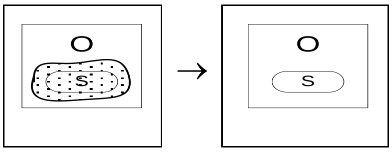


Figure E.8 — Creating an object state production

3) State removal: remove a *state* from an existing object, which is only possible if the *state* has no link to another *thing* in the OPD.

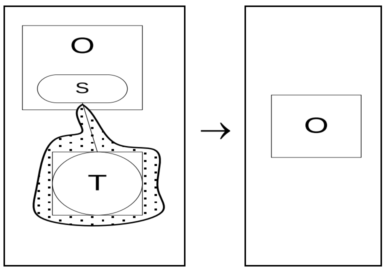


Figure E.9 — Removing an object state production

4) *Thing* removal: remove a *thing* from the OPD, which is only possible if the *thing* is not linked to another thing in the OPD.

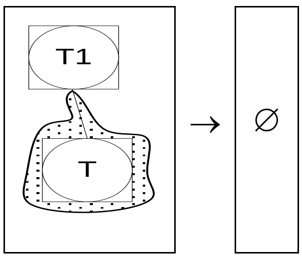


Figure E.10 — Removing an object production

5) Homogeneous structural link creation: these link productions connect *things* with the same persistence: Aggregation-Participation, Generalization-Specialization and Classification-Instantiation.

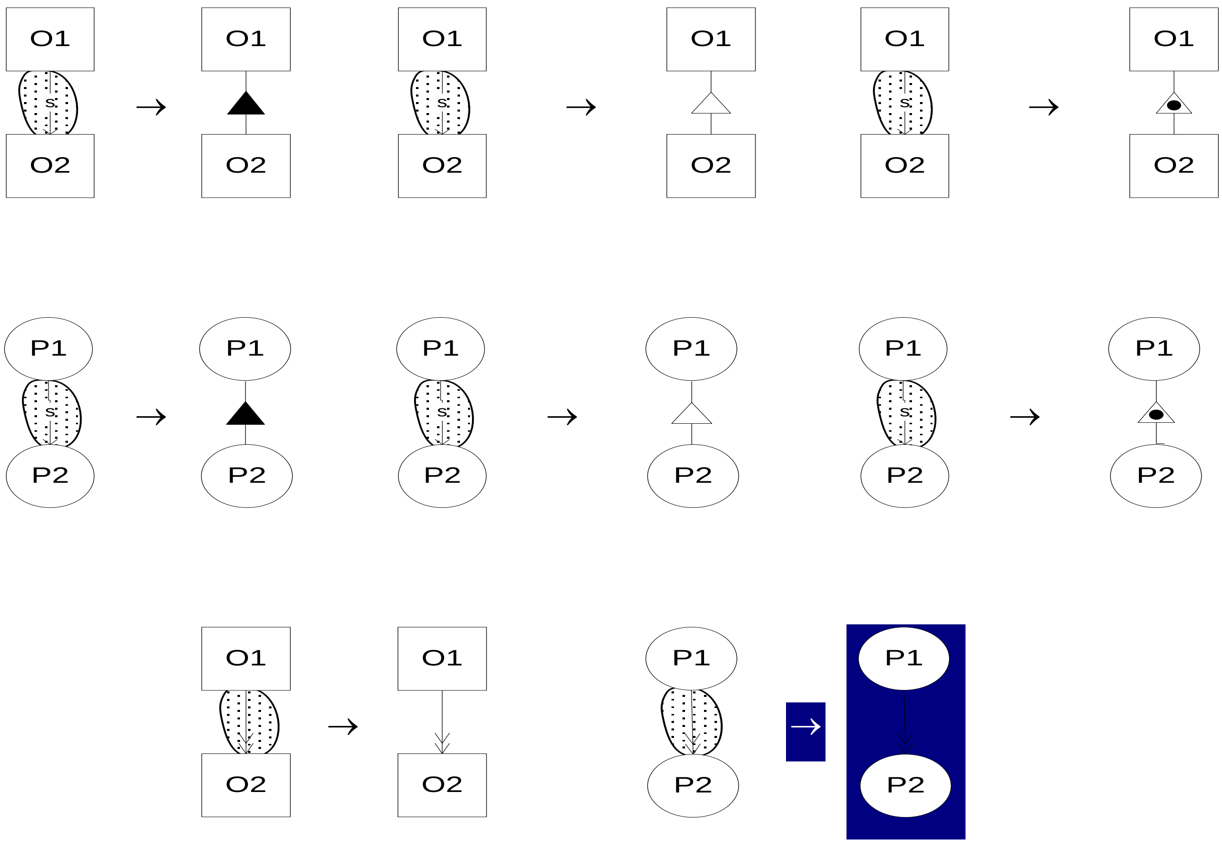


Figure E.11 — Structural link productions

6) Aggregation loop creation: the Aggregation-Participation link enables use to link an object to itself.

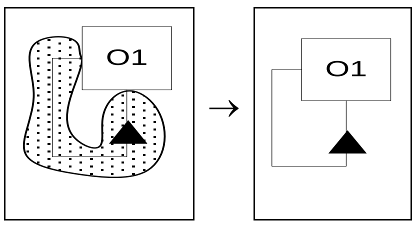


Figure E.12 — Aggregation loop link production

7) Generalization and Aggregation pair creation: the Aggregation-participation and the Generalization-Specialization link enables link co-exist together between two objects.

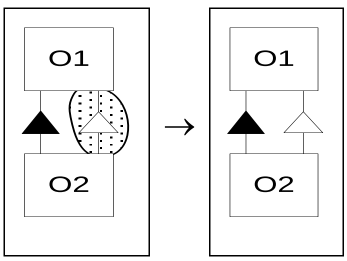


Figure E.13 — Generalization and aggregation pair creation production

8) Non-homogeneous structural link creation: the Exhibition-Characterization link enables the connection of two *things* of any persistence.

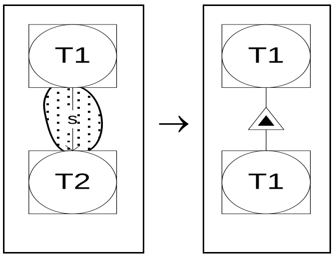


Figure E.14 — Production for Exhibition-Characterization link between things of same persistence

9) Object-to-Process link creation: create agent, instrument and consumption links between an object and a process.

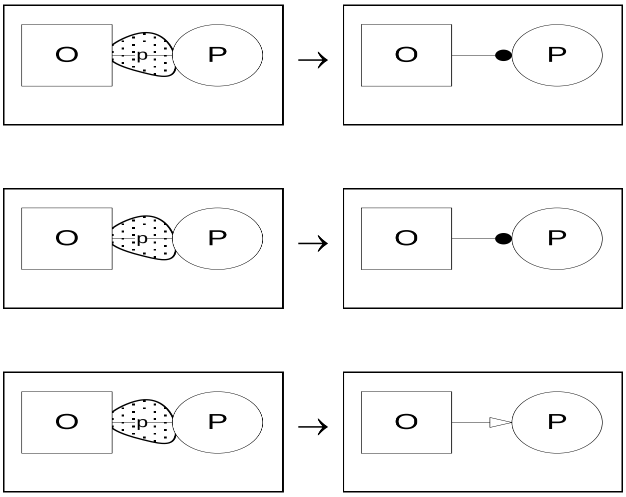


Figure E.15 — Object to process link creation production

10) Process-to-object link creation: create a result link between a process and an object.

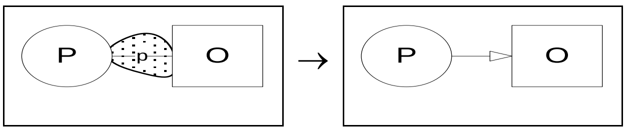


Figure E.16 — Process to object link creation production

11) Bi-directional procedure link creation: create an effect link between an object and a process.

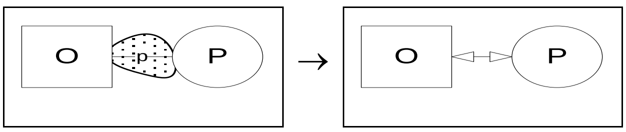


Figure E.17 — Bi-directional procedural link creation production

12) Invocation link creation: create an invocation link between two processes.

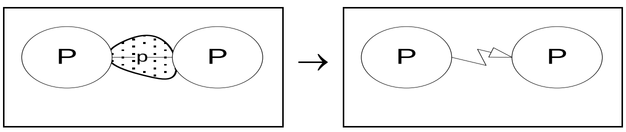


Figure E.18 — Invocation link creation production

13) Link removal: remove an existing link between two things.

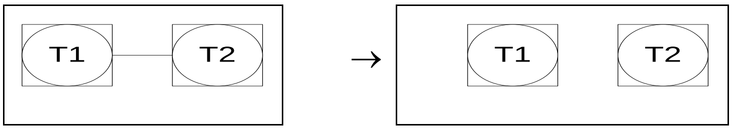


Figure E.19 — Link removal between two things

E.2.4 OPD validation

E.2.4.1 Validation overview

The validation of an existing OPD occurs by iteratively removing information from the OPD while maintaining its semantic validity (“abstracting” the OPD contents). Figure E.20 — Abstracting part consumption to effect on whole, depicts from left to right, an abstraction process abstracting details of O2.

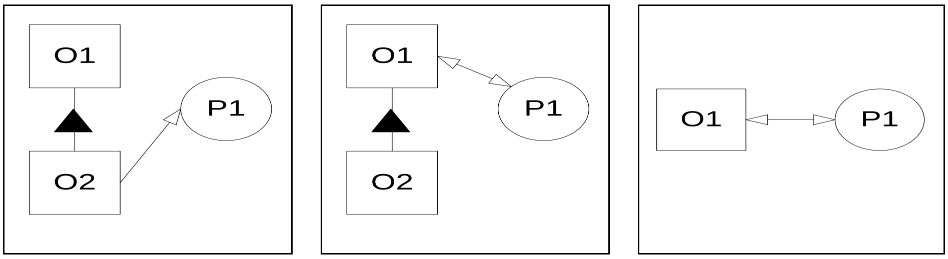


Figure E.20 — Abstracting part consumption to effect on whole

The left OPD shows that P1 consumes O2, which is a part of O1. By OPM semantics, this means that P1 changes O1, which is shown in the middle OPD. And finally, the removal of O2 reduces the amount of information in the OPD but maintains semantic validity.

During every abstraction step the validation algorithm checks for invalid constructs – a set of *elements* in the diagram that has invalid semantics. The diagram shown in Figure E.21 — An invalid link construction depicts an invalid construct, because while Process1 consumes Object2, its parent, Object1, which abstracts it, is linked to Process1 by only an agent link, which means that the process does not change the object.

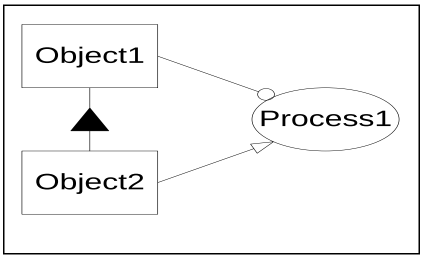


Figure E.21 — An invalid link construction

E.2.4.2 Validation algorithm

An OPD validation algorithm appears below. Since the number of abstraction productions and invalid constructs is very large, this Annex does not provide them all.

1) Calculate Type and Type Closure of all *things* in the OPD.

2) Validate all Process signatures by applying the Signature Consistency Validation algorithm. If validation failed, stop and return failure on signature validation.

3) While OPD contains *things* that have not been processed:

   i) Of all *things* in the current OPD select *thing* with max(height(thing)) and no outgoing structural links.

   ii) Transform all Temporary Links that start at *thing* to Regular Links.

   iii) Apply State Change Abstraction production to *thing* if applicable, as many times as possible.

   iv) Apply State-Specified Link Abstraction production to *thing* if applicable, as many times as possible.

   v) Apply Procedural Abstraction productions to *thing* if applicable, as many times as possible.

   vi) Check Illegal Constructs on *thing*. If illegal constructs exist, stop and return failure on *thing*.

   vii) Apply Thing Removal production to *thing* if applicable. If the production is not applicable, mark *thing* as processed.

4) Transform all temporary links in the OPD to regular links.

5) End.

E.2.4.3 Example ABS braking OPD abstraction

In this abstraction sequence, the ABS Ford system depicted in Figure E.22 — OPD for validation, reduces in detail to a less complicated OPD. The source OPD appears flattened to remove in-zooming.

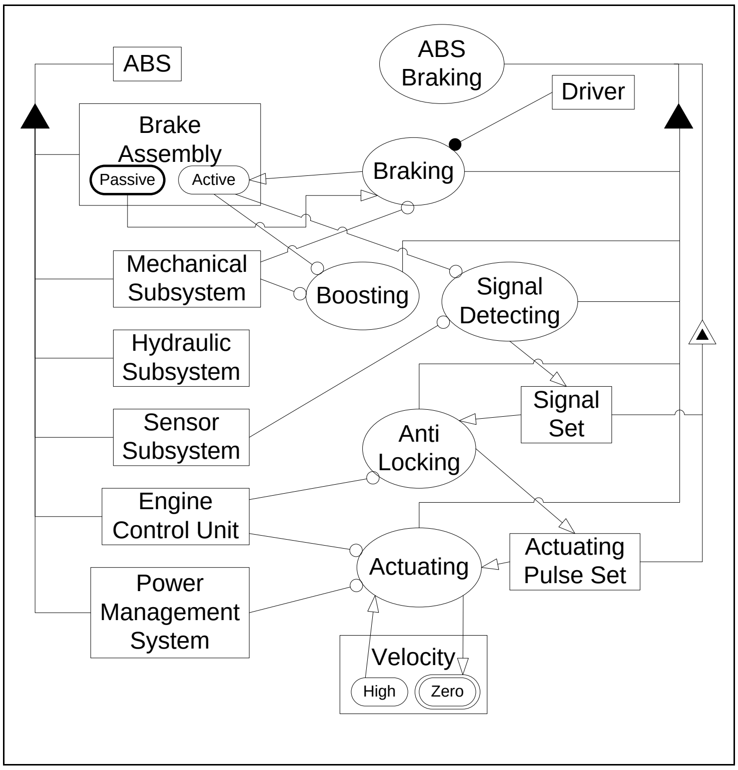


Figure E.22 — OPD for validation

Since the OPD for validation has no generalization or classification links in the diagram, the first step is to apply the abstraction steps. Check all the *objects* for removal of detail and then all the *processes*, beginning with *object* Brake Assembly.

The first task is to transform all temporary links. Since there are none, this step is complete. The next task is to apply State Change Abstraction to Brake Assembly using the link that starts at *state* Passive and ends at Braking, and the link that starts at Braking and ends at *state* Active as shown in Figure E.23 — State change abstraction. Since most of the remainder of the diagram remains the same, only the affected part appears.

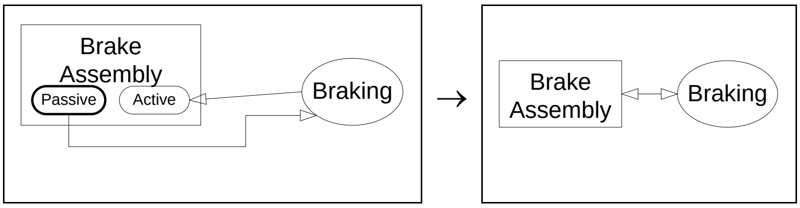


Figure E.23 — State change abstraction

The next task is to apply State-Specified Link abstraction. Two links begin at a state of Brake Assembly, one from *state* Active and ends at Boosting and the other from *state* Active and ends at Signal Detecting. The result of this task (once again removing unnecessary parts of the diagram) is shown in Figure E.24 — State-specified link abstraction.

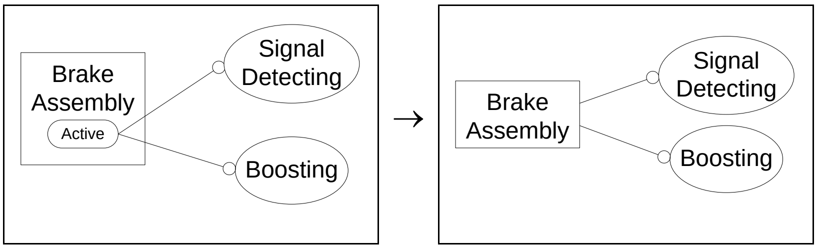


Figure E.24 — State-specified link abstraction

The next task is Procedural Abstraction. The procedural links that connect Brake Assembly to all other *things* in the diagram are "transferred" to its structural parent, which is ABS. The diagram then appears as shown, after removing the irrelevant *elements*, in Figure E.25 — Procedural abstraction.

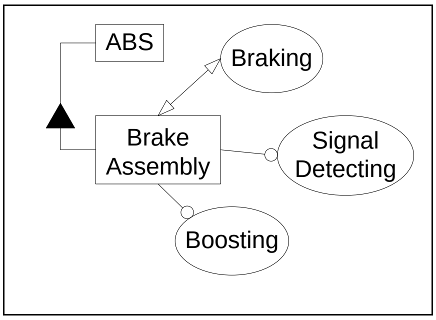


Figure E.25 — Procedural abstraction

The first link to abstract is the link to Boosting. The matching production for this case is Promotion of Part Instrument to Aggregate Instrument, as shown in Figure E.26 — Promotion of part instrument to aggregate instrument production.

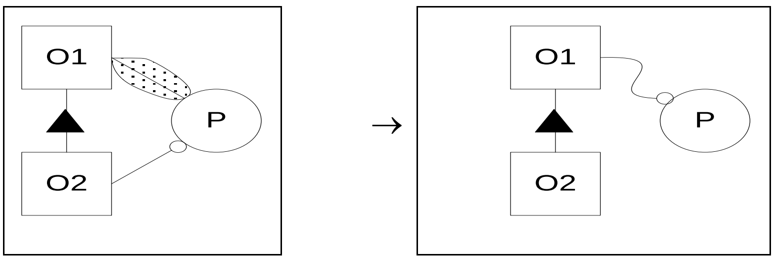


Figure E.26 — Promotion of part instrument to aggregate instrument production

Applying the production produces the diagram shown in Figure E.27 — Applying promotion production to Brake Assembly

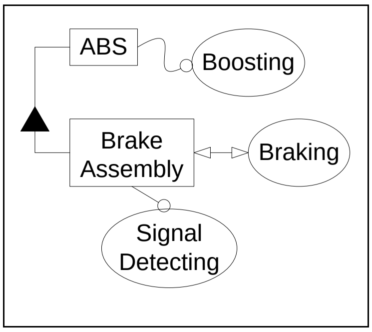


Figure E.27 — Applying promotion production to Brake Assembly

Using similar productions, the links from Brake Assembly to Braking and Signal Detecting create the diagram shown in Figure E.28 — Abstracting ABS links.

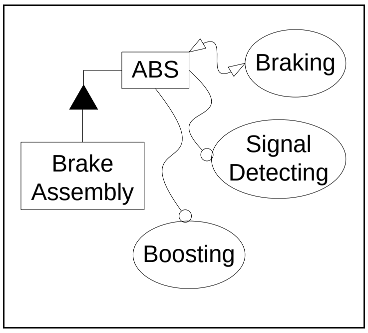


Figure E.28 — Abstracting ABS links

Since no illegal constructs are detected on Brake Assembly, the next task is Thing Removal. The result of the first round of the algorithm is shown in Figure E.29 — Removing disconnected things.

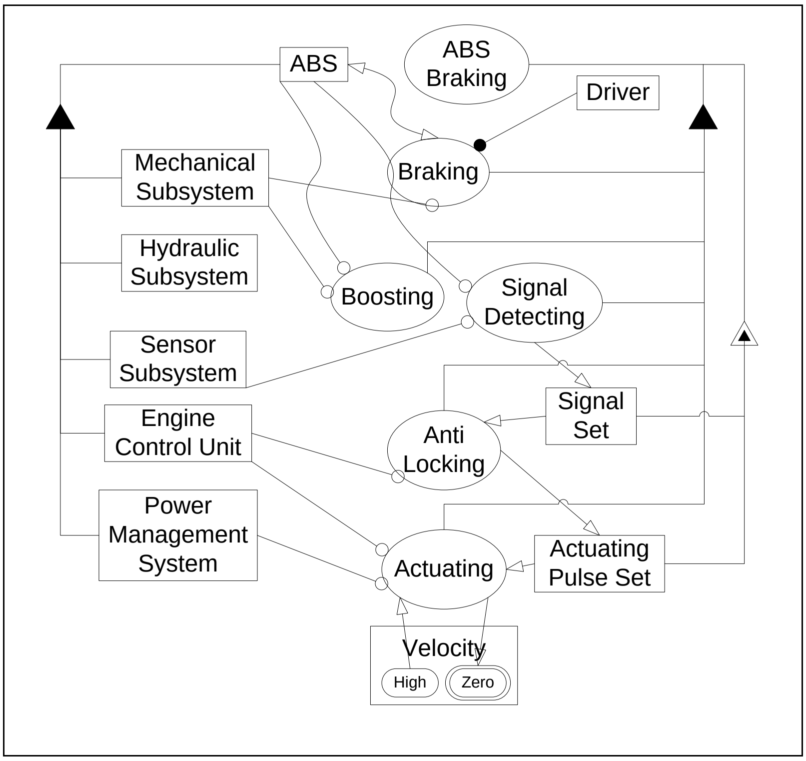


Figure E.29 — Removing disconnected things

A *process* is abstracted using the same steps used to abstract an *object*. The process Braking is abstracted next. After transforming the temporary links beginning at the *process*, the working diagram segment appears as Figure E.30 — Abstracting Braking process.

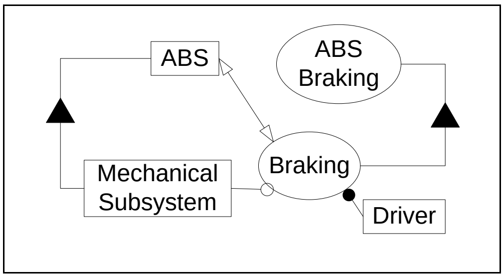


Figure E.30 — Abstracting Braking process

The tasks used to abstract a *process* are in general fewer than those used to abstract an *object* since a *process* does not contain states. Hence, the first task is Procedural Abstraction. After the application of the production, the diagram appears as shown in Figure E.31 — Procedural abstracting to ABS Braking.

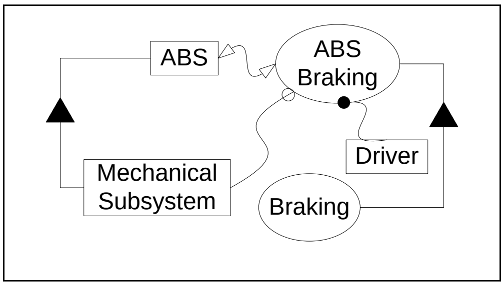


Figure E.31 — Procedural abstracting to ABS Braking

The next task is to remove Braking from the full diagram, yielding the diagram shown in Figure E.32 — Removing Braking from abstraction.

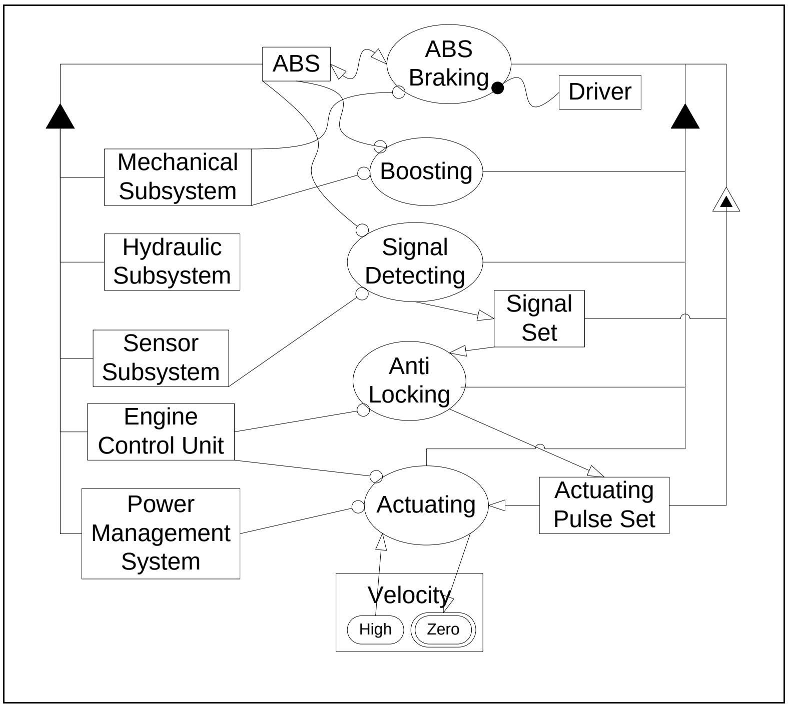


Figure E.32 — Removing Braking from abstraction

The abstraction process continues in the same way until there are no more *things* to abstract. Then, all the temporal links transform to regular links. The final diagram is shown in Figure E.33 — Final ABS Braking abstract process.

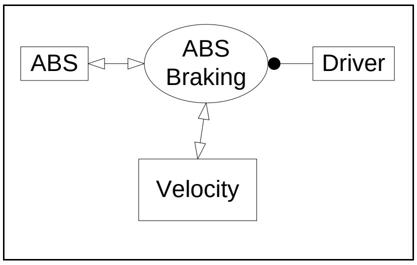


Figure E.33 — Final ABS Braking abstract process

Bibliography

[1] ISO/TC 184/SC 5. Terms of Reference: Study Group to Explore OPM for Modeling Standards, 2009. http://forums.nema.org:443/upload/N1049_OPM_Study_Group_Terms_of_Reference.doc

[2] ISO/TC 184/SC 5 N1070 Object Process Methodology Study Group – Interim Report 2010

[3] ISO/TC 184/SC 5 N1111 Object Process Methodology Study Group – Final Report 2011

[4] BIBLIOWICZ, A., A Graph Grammar-Based Formal Validation of an Object-Process Diagram, M. Sc. Thesis, Technion, Israel, 2008.

[5] BIBLIOWICZ, A., and DORI, D., A Graph Grammar-Based Formal Validation of Object-Process Diagrams. *Software and Systems Modeling*, 11, (2) pp. 287-302, 2012.

[6] CRAWLEY, E. F., MALMQVIST, J., ÖSTLUND, S., and BRODEUR, D. R., *Rethinking Engineering Education: The CDIO Approach*. Springer, 2007.

[7] DORI, D., Object-Process Methodology - A Holistic Systems Paradigm. Berlin : Springer Verlag, 2002

[8] DORI, D., Words from Pictures for Dual Channel Processing: A Bimodal Graphics-Text Representation of Complex Systems. *Communications of the ACM*, 51(5), pp. 47-52, 2008.

[9] DORI, D., FELDMAN, R., and STURM, A., From conceptual models to schemata: An object-process-based data warehouse construction method. *Information Systems* 33 (6), pp. 567-593, 2008.

[10] DORI, D., Object-Process Analysis: Maintaining the Balance between System Structure and Behavior. Journal of Logic and Computation, 5, 2, pp. 227-249, 1995.

[11] DORI, D., Object-Process Methodology – A Holistic Systems Paradigm, Springer Verlag, Berlin, Heidelberg, New York, 2002 (ISBN 3-540-65471-2; Foreword by Edward Crawley.

[12] DORI, D., REINHARTZ-BERGER, I. and STURM, A. Developing Complex Systems with Object-Process Methodology using OPCAT. LNCS 2813, pp. 570-572, 2003

[13] DORI, D., ViSWeb – The Visual Semantic Web: Unifying Human and Machine Knowledge Representations with Object-Process Methodology. *The International Journal on Very Large Data Bases (VLDB),* 13, 2, pp. 120-147, 2004.

[14] ESTEFAN, J., Survey of Model-Based Systems Engineering (MBSE) Methodologies 2 . Differentiating Methodologies from Processes, Methods, and Lifecycle Models. *Jet Propulsion*, *25*, 1–70, 2008. Retrieved from http://www.omgsysml.org/MBSE_Methodology_Survey_RevB.pdf

[15] GROBSHTEIN, Y. and DORI, D., Generating SysML Views from an OPM Model: Design and Evaluation. *Systems Engineering*, 14 (3), pp. 327-340, 2011.

[16] MYERSDORF, D., and DORI, D., The R&D Universe and Its Feedback Cycles: an Object-Process Analysis. *R&D Management*, 27, 4, pp. 333-344, 1997

[17] OLIVER, D. W., ANDARY, J. F., and FRISCH, H., Model-based systems engineering. In *Handbook of Systems Engineering and Management*, pp. 1361-1400, 2009.

[18] OSORIO, C. A., DORI, D., and SUSSMAN, J., COIM: An Object-Process Based Method for Analyzing Architectures of Complex, Interconnected, Large-Scale Socio-Technical Systems. *Systems Engineering* 14(3), 2011.

[19] PELEG, M., and DORI, D., The Model Multiplicity Problem: Experimenting with Real-Time Specification Methods. *IEEE Transaction on Software Engineering*, 26, 8, pp. 742-759, 2000.

[20] PELEG, M., SOMEKH, J., and DORI, D., A Methodology for Eliciting and Modeling Exceptions. *Journal of Biomedical Informatics* 42(4), pp. 736-747, 2009.

[21] OPCAT, Enterprise Systems Modeling Laboratory, Technion, Haifa, Israel, http://esml.iem.technion.ac.il/opm/

[22] RAMOS, A. L., FERREIRA, J. V., BARCELÓ, J., LITHE: An Agile Methodology for Human-Centric Model-Based Systems Engineering. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 2012.

[23] REICHWEIN, A., and PAREDIS, C., Overview of Architecture Frameworks and Modeling Languages for Model-Based Systems Engineering. *Proceedings of the ASME 2011 International Design Engineering Technical Conferences Computers and Information in Engineering Conference*, 1-9, 2011.

[24] REINHARTZ-BERGER, I., and DORI, D., A Reflective Metamodel of Object-Process Methodology: The System Modeling Building Blocks. In Business Systems Analysis with Ontologies, P. Green and M. Rosemann (Eds.), Idea Group, Hershey, PA, USA, pp. 130-173, 2005.

[25] SHARON, A., de WECK, O. and DORI, D., Model-Based Design Structure Matrix: Deriving a DSM from an Object-Process Model. *Systems Engineering*, pp. 1-14, 2012.

[26] SOMEKH, J., CHODER, M., and DORI, D., Conceptual Model-Based Systems Biology: Mapping Knowledge and Discovering Gaps in the mRNA Transcription Cycle. *PLoS ONE*, 7(12): e51430. doi:10.1371/journal.pone.0051430, Dec. 20, 2012.

[27] SOFFER, P., GOLANY, B., and DORI, D., ERP Modeling: A Comprehensive Approach. *Information Systems* 28, 6, pp. 673-690, 2003.

[28] STURM, A., DORI, D., and SHEHORY, O., An Object-Process-Based Modeling Language for Multi-Agent Systems. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, 40 (2) pp. 227-241, 2010.

[29] STURM, A., DORI, D., and SHEHORY, O., Application-Based Domain Analysis Approach and Its Object-Process Methodology Implementation. *International Journal of Software Engineering and Knowledge Engineering*, 19, 1, February 2009.

[30] YAROKER, Y., PERELMAN, V., and DORi, D., An OPM Conceptual Model-Based Executable Simulation Environment: Implementation and Evaluation. *Systems Engineering*, 16(4), pp. 381-390, 2013.

