

PUBLICLY
AVAILABLE
SPECIFICATION

ISO/PAS
19450

First edition
2015-12-15

Automation systems and integration — Object-Process Methodology

*Systèmes d'automatisation et intégration — Object-Process
Methodology*



Reference number
ISO/PAS 19450:2015(E)

AVA Documents
Document Sharing Hub
© ISO 2015



COPYRIGHT PROTECTED DOCUMENT

© ISO 2015, Published in Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Contents

Page

Foreword	vi
Introduction	vii
1 Scope	1
2 Normative references	1
3 Terms and definitions	1
4 Symbols	8
5 Conformance	10
6 OPM principles and concepts	10
6.1 OPM modelling principles	10
6.1.1 Modelling as a purpose-serving activity	10
6.1.2 Unification of function, structure, and behaviour	11
6.1.3 Identifying functional value	11
6.1.4 Function versus behaviour	11
6.1.5 System boundary setting	12
6.1.6 Clarity and completeness trade-off	12
6.2 OPM Fundamental concepts	12
6.2.1 Bimodal representation	12
6.2.2 OPM modelling elements	12
6.2.3 OPM things: objects and processes	13
6.2.4 OPM links: procedural and structural	13
6.2.5 OPM context management	14
6.2.6 OPM model implementation	14
7 OPM thing syntax and semantics	15
7.1 Objects	15
7.1.1 Description	15
7.1.2 Representation	15
7.2 Processes	15
7.2.1 Description	15
7.2.2 Representation	16
7.3 OPM things	16
7.3.1 OPM thing defined	16
7.3.2 Object-process test	16
7.3.3 OPM thing generic properties	17
7.3.4 Default values of thing generic properties	18
7.3.5 Object states	18
8 OPM link syntax and semantics overview	20
8.1 Procedural link overview	20
8.1.1 Kinds of procedural links	20
8.1.2 Procedural link uniqueness OPM principle	20
8.1.3 State-specified procedural links	20
8.2 Operational semantics and flow of execution control	20
8.2.1 The Event-Condition-Action control mechanism	20
8.2.2 Preprocess object set and postprocess object set	21
8.2.3 Skip semantics of condition versus wait semantics of non-condition links	21
9 Procedural links	22
9.1 Transforming links	22
9.1.1 Kinds of transforming links	22
9.1.2 Consumption link	22
9.1.3 Result link	23
9.1.4 Effect link	23
9.1.5 Basic transforming links summary	23

9.2	Enabling links.....	24
9.2.1	Kinds of enabling links	24
9.2.2	Agent and Agent Link.....	24
9.2.3	Instrument and Instrument Link.....	25
9.2.4	Basic enabling links summary.....	26
9.3	State-specified transforming links.....	26
9.3.1	State-specified consumption link.....	26
9.3.2	State-specified result link.....	27
9.3.3	State-specified effect links.....	28
9.3.4	State-specified transforming links summary.....	30
9.4	State-specified enabling links	31
9.4.1	State-specified agent link.....	31
9.4.2	State-specified instrument link.....	32
9.4.3	State-specified enabling links summary	32
9.5	Control links.....	33
9.5.1	Kinds of control links	33
9.5.2	Event links	34
9.5.3	Condition links.....	40
9.5.4	Exception links	47
10	Structural links	48
10.1	Kinds of structural links.....	48
10.2	Tagged structural link.....	48
10.2.1	Unidirectional tagged structural link.....	48
10.2.2	Unidirectional null-tagged structural link.....	49
10.2.3	Bidirectional tagged structural link	49
10.2.4	Reciprocal tagged structural link.....	49
10.3	Fundamental structural relations.....	50
10.3.1	Kinds of fundamental structural relations	50
10.3.2	Aggregation-participation relation link.....	51
10.3.3	Exhibition-characterization link.....	52
10.3.4	Generalization-specialization and inheritance.....	55
10.3.5	Classification-instantiation link.....	58
10.3.6	Fundamental structural relation link and tagged structural link summary.....	61
10.4	State-specified structural relations and links	62
10.4.1	State-specified characterization relation link.....	62
10.4.2	State-specified tagged structural relations	63
11	Relationship cardinalities	67
11.1	Object multiplicity in structural and procedural links	67
11.2	Object multiplicity expressions and constraints.....	69
11.3	Attribute value and multiplicity constraints.....	71
12	Logical operators: AND, XOR, and OR.....	71
12.1	Logical AND procedural links.....	71
12.2	Logical XOR and OR procedural links.....	73
12.3	Diverging and converging XOR and OR links	74
12.4	State-specified XOR and OR link fans	76
12.5	Control-modified link fans	77
12.6	State-specified control-modified link fans	77
12.7	Link probabilities and probabilistic link fans.....	79
13	Execution path and path labels	81
14	Context management with OPM.....	83
14.1	Completing the SD.....	83
14.2	Achieving model comprehension.....	83
14.2.1	OPM refinement-abstraction mechanisms.....	83
14.2.2	Control (operational) semantics within an in-zoomed process context	87
14.2.3	OPM fact consistency principle.....	98
14.2.4	Abstraction ambiguity resolution for procedural links.....	99

Annex A (normative) OPL formal syntax in EBNF	102
Annex B (informative) Guidance for OPM	121
Annex C (informative) Modelling OPM using OPM	124
Annex D (informative) OPM dynamics and simulation	157
Bibliography	163

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.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

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. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: [Foreword - Supplementary information](#)

The committee responsible for this document is Technical Committee ISO/TC 184, *Automation systems and integration*, Subcommittee SC 5, *Interoperability, integration, and architectures for enterprise systems and automation applications*.

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 Publicly Available Specification 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 the syntax and semantics of OPL 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 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 Publicly Available Specification provides a presentation of the normative text that follows the Extended Backus-Naur Form (EBNF) specification of the language syntax. All elements are presented in [Clauses 6 to 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. Clause 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.

The International Organization for Standardization (ISO) 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 modelling system given in [Clauses 6](#) to [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 specifies Object-Process Methodology (OPM) with detail sufficient for enabling practitioners to utilise the concepts, semantics, and syntax of Object-Process Methodology 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 Publicly Available Specification presents some examples for the use of Object-Process Methodology to improve clarity, it does not attempt to provide a complete reference for all the possible applications of Object-Process Methodology.

2 Normative references

There are no normative references.

3 Terms and definitions

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

3.1

abstraction

decreasing the extent of detail and system model *completeness* (3.8) in order to achieve better comprehension

3.2

affectee

transformee (3.78) that is affected by a *process* (3.58) occurrence, i.e. its *state* (3.69) changes

Note 1 to entry: An affectee can only be a *stateful object* (3.66). A *stateless object* (3.67) can only be created or consumed, but not affected.

3.3

agent

enabler (3.17) that is a human or a group of humans

3.4

attribute

object (3.39) that characterizes a *thing* (3.76) other than itself

3.5

behaviour

transformation (3.77) of *objects* (3.39) resulting from the execution of an *Object-Process Methodology* (3.43) model comprising a collection of *things* (3.76) and *links* (3.36) to objects in the model

3.6

beneficiary

<system> *stakeholder* (3.65) who gains functional *value* (3.82) from the system's *operation* (3.46)

3.7

class

collection of *things* (3.76) with the same *perseverance* (3.50), essence, and affiliation values, and the same *feature* (3.21) and *state* (3.69) set

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 (3.56) from an *object* (3.39) or *object state* (3.69) to a *process* (3.58), denoting a procedural constraint

3.10

consume

transformee (3.78) that a *process* (3.58) occurrence consumes or eliminates

3.11

context

<model> portion of an *Object-Process Methodology* (3.43) model represented by an *Object-Process Diagram* (3.41) and corresponding *Object-Process Language* (3.42) text

3.12

control link

procedural link (3.56) with additional control semantics

3.13

control modifier

symbol embellishing a *link* (3.36) to add control semantics to it, making it a *control link* (3.12)

Note 1 to entry: The control modifiers are the symbols 'e' for *event* (3.18) and 'c' for condition.

3.14

discriminating attribute

attribute (3.4) whose different *values* (3.81) identify corresponding specialization relations

3.15

effect

change in the *state* (3.69) of an *object* (3.39) or an *attribute* (3.4) *value* (3.81)

Note 1 to entry: An effect only applies to a *stateful object* (3.66).

3.16

element

thing (3.76) or *link* (3.36)

3.17

enabler

<process> *object* (3.39) that enables a *process* (3.58) 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 (3.39) to a particular *state* (3.69), either of which may initiate an evaluation of the *process* (3.58) *precondition* (3.53)

3.19

event link

control link (3.12) denoting an *event* (3.18) originating from an *object* (3.39) or *object state* (3.69) to a *process* (3.58)

3.20**exhibitor**

thing (3.76) that exhibits (is characterized by) a *feature* (3.21) by means of the exhibition-characterization relation

3.21**feature**

attribute (3.4) or *operation* (3.46)

3.22**folding**

mechanism of *abstraction* (3.1) achieved by hiding the *refineables* (3.61) of an unfolded *refinee* (3.62)

Note 1 to entry: The four kinds of folded refineables are parts (part folding), *features* (3.21) (feature folding), specializations (specialization folding), and *instances* (3.28) (instance folding).

Note 2 to entry: Folding is primarily applied to *objects* (3.39). 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 to entry: The opposite of folding is *unfolding* (3.80).

3.23**function**

process (3.58) that provides functional *value* (3.82) to a *beneficiary* (3.6)

3.24**general**

<OPM> *refineable* (3.61) with specializations

3.25**informatical**

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

3.26**inheritance**

assignment of *Object-Process Methodology* (3.43) *elements* (3.16) of a *general* (3.24) to its specializations

3.27**input link**

link (3.36) from *object* (3.39) source (input) *state* (3.69) to the transforming *process* (3.58)

3.28**instance**

<model> *object* (3.39) instance or *process* (3.58) instance that is a *refinee* (3.62) in a classification-instantiation relation

3.29**instance**

<operational> *object* (3.39) instance or *process* (3.58) instance that is an actual, uniquely identifiable *thing* (3.76) that exists during model *operation* (3.46), e.g. during simulation or runtime implementation

Note 1 to entry: A process instance is identifiable by the operational instances of the *involved object set* (3.32) during process occurrence and the process start and end time stamps of the occurrence.

3.30**instrument**

non-human *enabler* (3.17)

3.31**invocation**

<process> initiating of a *process* (3.58) by a process

3.32

involved object set

union of *preprocess object set* ([3.54](#)) and *postprocess object set* ([3.52](#))

3.33

in-zoom context

things ([3.76](#)) and *links* ([3.36](#)) within the boundary of the thing being *in-zoomed*

3.34

in-zooming

<object> *object* ([3.39](#)) part *unfolding* ([3.80](#)) that indicates spatial ordering of the constituent objects

3.35

in-zooming

<process> *process* ([3.58](#)) part *unfolding* ([3.80](#)) that indicates temporal partial ordering of the constituent processes

3.36

link

graphical expression of a *structural relation* ([3.73](#)) or a *procedural relation* ([3.57](#)) between two *Object-Process Methodology* ([3.43](#)) *things* ([3.76](#))

3.37

metamodel

model of a modelling language or part of a modelling language

3.38

model fact

relation between two *Object-Process Methodology* ([3.43](#)) *things* ([3.76](#)) or *states* ([3.69](#)) in the Object-Process Methodology model

3.39

object

<OPM> model *element* ([3.16](#)) representing a *thing* ([3.76](#)) that does or might exist physically or *informatically* ([3.25](#))

3.40

object class

pattern for *objects* ([3.39](#)) that have the same *structure* ([3.74](#)) and pattern of *transformation* ([3.77](#))

3.41

Object-Process Diagram

OPD

Object-Process Methodology ([3.43](#)) graphic representation of an Object-Process Methodology model or part of a model, in which *objects* ([3.39](#)) and *processes* ([3.58](#)) in the universe of interest appear together with the *structural links* ([3.72](#)) and *procedural links* ([3.56](#)) among them

3.42

Object-Process Language

OPL

subset of English natural language that represents textually the *Object-Process Methodology* ([3.43](#)) model that the *Object-Process Diagram* ([3.42](#)) 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* ([3.39](#)) in the system and their *transformation* ([3.77](#)) or use by *processes* ([3.58](#))

3.44**OPD object tree**

tree graph, whose root is an *object* (3.39), depicting elaboration of the object through *refinement* (3.63)

3.45**OPD process tree**

tree graph whose root is the *System Diagram* (3.75) and each node is an *Object-Process Diagram* (3.42) obtained by *in-zooming* (3.35) of a *process* (3.58) in its ancestor Object-Process Diagram (or the System Diagram) and each directed edge points from the refined process at the parent Object-Process Diagram to the same process in the child Object-Process Diagram

Note 1 to entry: *Object-Process Methodology* (3.43) model elaboration usually occurs by process decomposition through in-zooming, therefore the OPD process tree is the primary way to navigate an Object-Process Methodology model.

3.46**operation**

process (3.58) that a *thing* (3.76) performs, which characterizes the thing other than itself

3.47**output link**

link (3.36) from the transforming *process* (3.58) to the output (destination) *state* (3.69) of an *object* (3.39)

3.48**out-zooming**

<object> inverse of *object* (3.39) *in-zooming* (3.34)

3.49**out-zooming**

<process> inverse of *process* (3.58) *in-zooming* (3.35)

3.50**perseverance**

property (3.60) of *thing* (3.76) which can be static, defining an *object* (3.39), or dynamic, defining a *process* (3.58)

3.51**postcondition**

<process> condition that is the outcome of successful *process* (3.58) completion

3.52**postprocess object set**

collection of *objects* (3.39) remaining or resulting from *process* (3.58) completion

Note 1 to entry: The postprocess object set may include *stateful objects* (3.66), for which specific *states* (3.69) result from process performance.

3.53**precondition**

<process> condition for starting a *process* (3.58)

3.54**preprocess object set**

collection of *objects* (3.39) to evaluate prior to starting a *process* (3.58)

Note 1 to entry: The collection of the objects may include *stateful objects* (3.66) for which specific *states* (3.69) are necessary for process performance.

3.55**primary essence**

<system> *essence* of the majority of *things* (3.76) in a system, which can be either *informatical* (3.25) or physical

3.56

procedural link

graphical notation of *procedural relation* ([3.57](#)) in *Object-Process Methodology* ([3.43](#))

3.57

procedural relation

connection or association between an *object* ([3.39](#)) or object *state* ([3.69](#)) and a *process* ([3.58](#))

Note 1 to entry: Procedural relations specify how the system operates to attain its *function* ([3.23](#)), designating time-dependent or conditional initiating of processes that transform objects.

Note 2 to entry: An *invocation* ([3.31](#)) or exception *link* ([3.36](#)) signifies a transient object in the flow of execution control between two processes.

3.58

process

transformation ([3.77](#)) of one or more *objects* ([3.39](#)) in the system

3.59

process class

pattern for *processes* ([3.58](#)) that perform the same *object* ([3.39](#)) *transformation* ([3.77](#)) pattern

3.60

property

modelling annotation common to all *elements* ([3.16](#)) of a specific kind that serve to distinguish that *element*

Note 1 to entry: Cardinality constraints, path labels, and *structural link* ([3.72](#)) tags are frequent property annotations.

Note 2 to entry: Unlike an *attribute* ([3.4](#)), 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 to entry: Property is an attribute of an element in the *Object-Process Methodology* ([3.43](#)) *metamodel* ([3.37](#)).

3.61

refineable

<OPM> *thing* ([3.76](#)) amenable to *refinement* ([3.63](#)), which can be a *whole* ([3.83](#)), an *exhibitor* ([3.20](#)), a *general* ([3.24](#)), or a *class* ([3.7](#))

3.62

refinee

thing ([3.76](#)) that refines a *refineable* ([3.61](#)), which can be a part, a *feature* ([3.21](#)), a specialization, or an instance ([3.29](#))

Note 1 to entry: 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* ([3.8](#))

3.64

resultee

transformee ([3.78](#)) that a *process* ([3.58](#)) 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 (3.39) with specified *states* (3.69)

3.67**stateless object**

object (3.39) lacking specified *states* (3.69)

3.68**state**

<object> possible situation or position of an *object* (3.39)

Note 1 to entry: In *Object-Process Methodology* (3.43) there is no concept of *process* (3.58) *state*, such as “started”, “in process”, or “finished” within a model. Instead, Object-Process Methodology represents and models subprocesses, such as starting, processing, or finishing. See also discussion of Object-Process Methodology 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* (3.39) instances, current states of each *stateful object* (3.66) instance, and the *process* (3.58) instances, with their elapsed times, executing at the time the snapshot occurs

3.70**state expression**

refinement (3.63) involving the revealing of any proper subset of an *object's* (3.39) set of *states* (3.69)

3.71**state suppression**

abstraction (3.1) involving the hiding of any proper subset of an *object's* (3.39) set of *states* (3.69)

3.72**structural link**

graphic notation of *structural relation* (3.73) in *Object-Process Methodology* (3.43)

3.73**structural relation**

operationally invariant connection or association between things

Note 1 to entry: 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* (3.39) in an *Object-Process Methodology* (3.43) model and the non-transient relations or associations among them

3.75**System Diagram****SD**

Object-Process Diagram (3.41) with one systemic *process* (3.58) indicating the system *function* (3.23) and the *objects* (3.39) connecting with that function to depict the overall *context* (3.11) for and top-level view of the system

Note 1 to entry: System Diagram is the root of the *OPD process tree* (3.45) and has no extent of detail beyond the overall context depicted, i.e. no in-zoomed *refinee* (3.62) is present. Any Object-Process Diagram other than System Diagram is a node in the OPD process tree resulting from *refinement* (3.63).

3.76**thing**

<OPM> *object* (3.39) or *process* (3.58)

3.77

transformation

creation (generation, construction) or consumption (elimination, destruction) of an *object* (3.39) or a change in the *state* (3.69) of an object

Note 1 to entry: Only a *process* (3.58) can perform transformation.

3.78

transformee

object (3.39) that a *process* (3.58) transforms (creates, consumes, or affects)

3.79

transforming link

consumption link, effect link, or result link

3.80

unfolding

refinement (3.63) that elaborates a *refinee* (3.62) with additional detail comprising other *things* (3.76) and the *links* (3.36) between them.

Note 1 to entry: The four kinds of unfolding are part unfolding, feature unfolding, specialization unfolding, and instance unfolding.

Note 2 to entry: Unfolding is primarily applied to *objects* (3.39) for exposing details about the unfolded object.

3.81

value

<attribute> *state* (3.69) of an *attribute* (3.4)

3.82

value

<functional> benefit at cost that the system's *function* (3.23) delivers

3.83

whole

aggregate *thing* (3.76) comprised of two or more parts, each having the same *perseverance* (3.50) as the aggregate

4 Symbols



object



physical object



environmental object



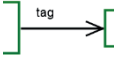
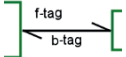
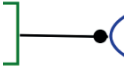
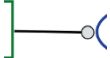
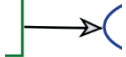
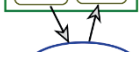
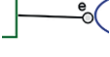
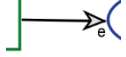
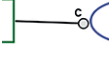
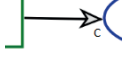
process



physical process

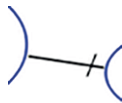


environmental process

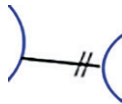
	state
	aggregation-participation
	exhibition-characterization
	generalization-specialization
	classification-instantiation
	unidirectional tagged structural link
	bidirectional tagged structural link
	agent link
	instrument link
	effect link
	consumption link
	result link
	input-output link pair
	instrument event link
	consumption event link
	instrumental condition link
	consumption condition link
	invocation link



self-invocation link



over-time exception link



under-time exception link

5 Conformance

Anticipating that the implementation of this Publicly 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 OPM shall use the language part of OPM, namely OPM Semantics and Syntax, by:
 - 1) using only OPM symbols defined in [Clause 4](#) with the meaning assigned to them in this Publicly Available Specification; and,
 - 2) using only OPM elements defined in [Clauses 7](#) to [12](#) with the meaning assigned to them in this Publicly Available Specification.
- b) Full conformance with OPM shall require:
 - 1) conformance with a); and,
 - 2) conformance with the approach and scheme of modelling systems with OPM, as defined in [Clauses 6](#) and [14](#).
- c) Conformance by toolmakers shall require:
 - 1) conformance with a);
 - 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 Annex A.

6 OPM 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 Identifying 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 OPM 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 OPD, shall have an equivalent OPM textual paragraph comprised of one or more OPM language sentences using the 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 14977:1996.

NOTE 3 For most of the OPD figures in this Publicly Available Specification, 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 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](#) provides an OPM metamodel overview.

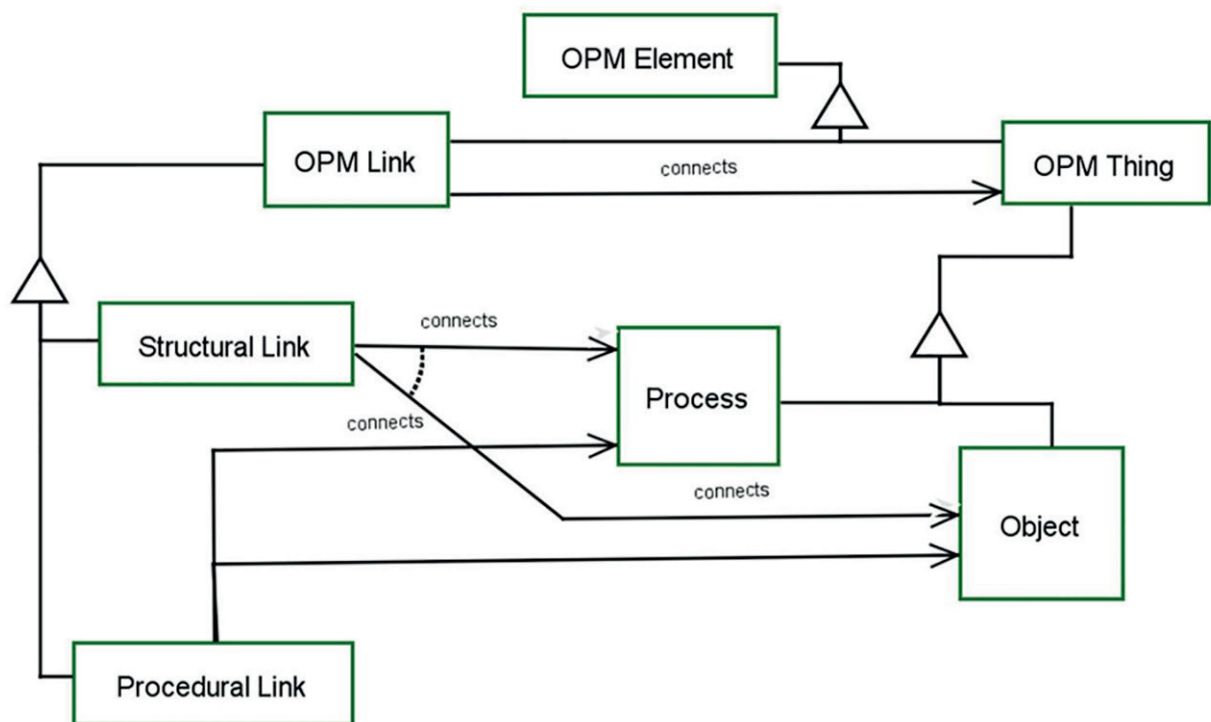


Figure 1 — OPM metamodel overview

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 structure and process decomposition 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 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. Thing names shall be unique within that management context.

6.2.6 OPM model implementation

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 Publicly 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 Publicly 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 producing 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 corresponding 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 Conventions for naming objects are discussed in [B.6.2](#).

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 can directly invoke another process, by means of the invocation link (see 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 maintain 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 Conventions for naming processes are discussed in B.6.3.

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 OPM.

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:

- it has a time association;
- it associates with the verb to fly; and
- it transforms **Airplane** by changing the value of its location attribute from source to destination.

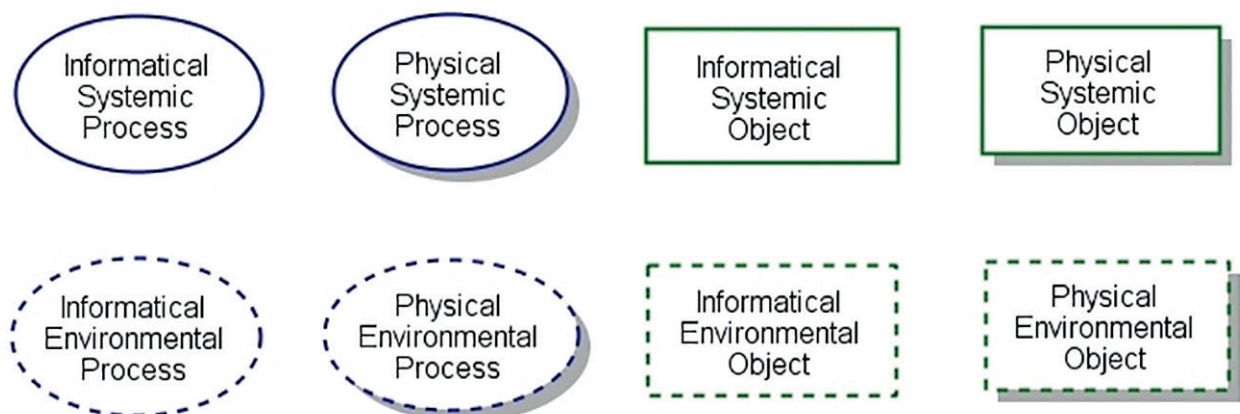
7.3.3 OPM thing generic properties

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. Accordingly, the permissible value for the **Perseverance** property is static or dynamic.
- **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.

NOTE 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 9.5.2.5.1), can exist.

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 can 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.

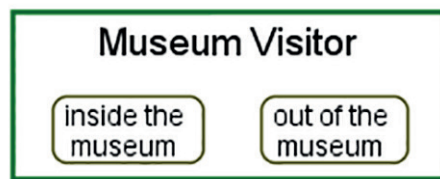
NOTE 1 Depending upon model behaviour, operational instances of an object can be at different states.

NOTE 2 Conventions for naming object states are discussed in [B.6.4](#).

7.3.5.2 Object state representation

Graphically, a labelled, rounded-corner rectangle (a 'routangle') 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 can 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 can 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 can have with respect to the process to which they link. Hence, an object can 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, Do 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 and action occurs.

Starting performance of a process has two prerequisites:

- a) an initiating event, and
- b) 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 or 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. Consumeers are only members of the preprocess object set, while resultees are only members of the postprocess object set.

NOTE 2 The operational instance semantics for objects in the involved object set are presented in [14.2.2.4.4](#).

8.2.3 Skip semantics of condition versus 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 can 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 can 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 that changes 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.

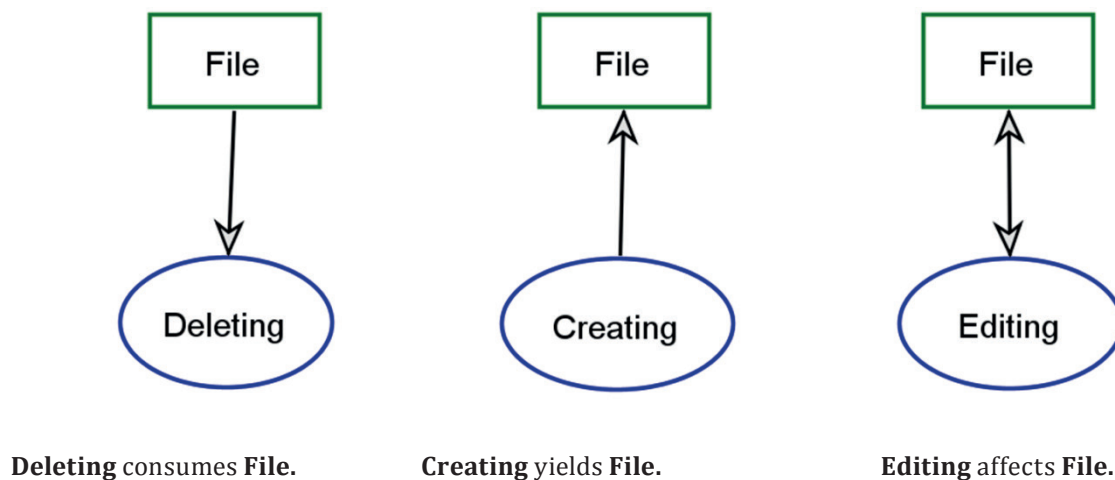


Figure 7 — Transforming links: left – consumption, middle – result, right – effect

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 **Consume**.

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 can 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** in litres with value **1000** and the consumption link has a property **Flow Rate** in litres per second with value **50**. In this case, if **Irrigating** is uninterrupted, it will last 20 s, 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** (m³). The **Refining** to **Gasoline** result link has the property **Gasoline Yield Rate** (m³/h) with value **1000** and the **Refining** to **Diesel Oil** result link has the property **Diesel Oil Yield Rate** (m³/h) with value **800**. Assuming there is enough **Crude Oil**, if **Refining** activates and performs for 10 h, it will yield 10 000 m³ of **Gasoline** and 8 000 m³ 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.

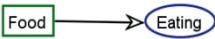
Graphically, a bidirectional arrow with two closed arrowheads, as shown in [Figure 7](#), one pointing in each direction between the affecting process and the affected object shall denote the effect link.

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

9.1.5 Basic transforming links summary

[Table 1](#) summarizes the basic transforming links.

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 can 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 can model premature process ending without completion as exception handling (see 9.5.4).

9.2.2 Agent and Agent Link

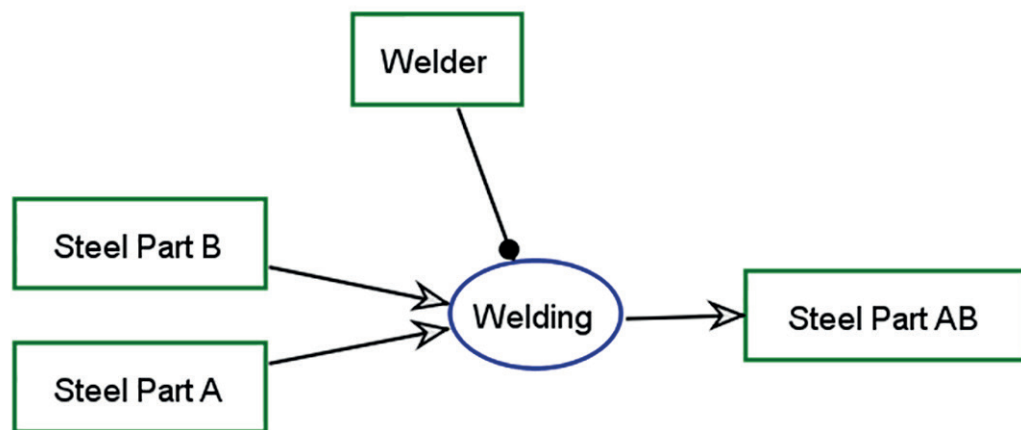
An agent shall be a human or a group of humans capable of intelligent decision-making, who interact with the system to enable or control the process throughout performance of the process.

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.

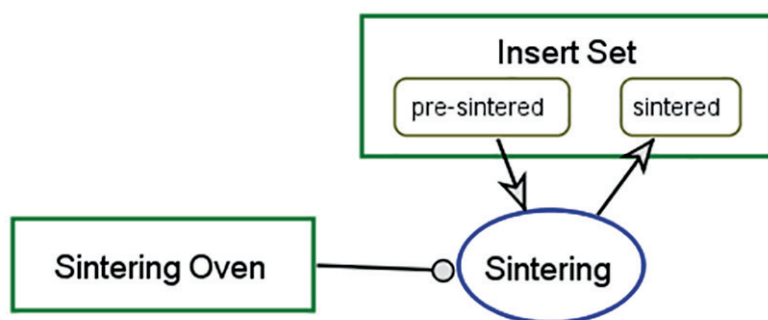
An instrument link shall be an enabling link from the instrument object to the process it enables, specifying that the instrument object is necessary for linked process activation and performance.

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 might 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 performing 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 summarizes the basic enabling links.

Table 2 — Basic 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 consume and the consume shall have an attribute that indicates the available quantity.

NOTE 1 The modeller can 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 consume 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 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 can 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 consume 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 might want the OPL on the right in [Figure 10](#), but the OPL on the left reduces ambiguity.

EXAMPLE 2

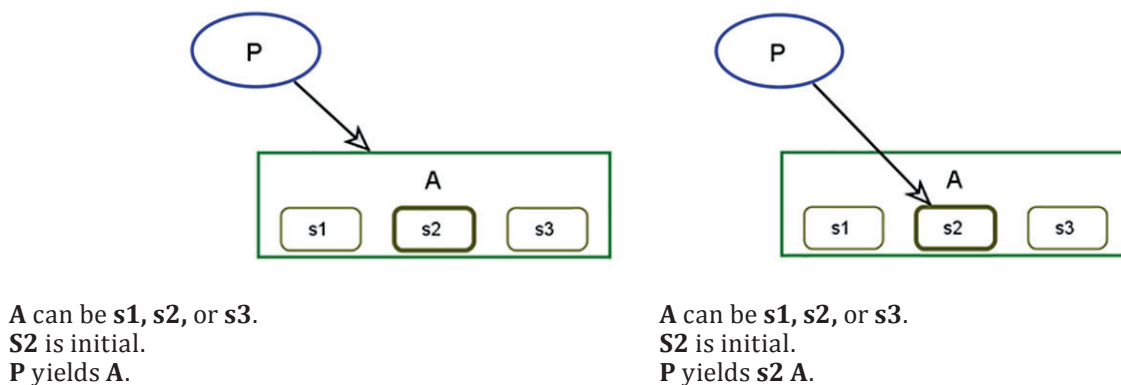


Figure 10 — Correct (left) and incorrect (right) result link to an object with an initial state

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:

- a) input-output-specified effect link specifying both input source and output destination states;
- b) input-specified effect link specifying only the input source state; and
- c) 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.

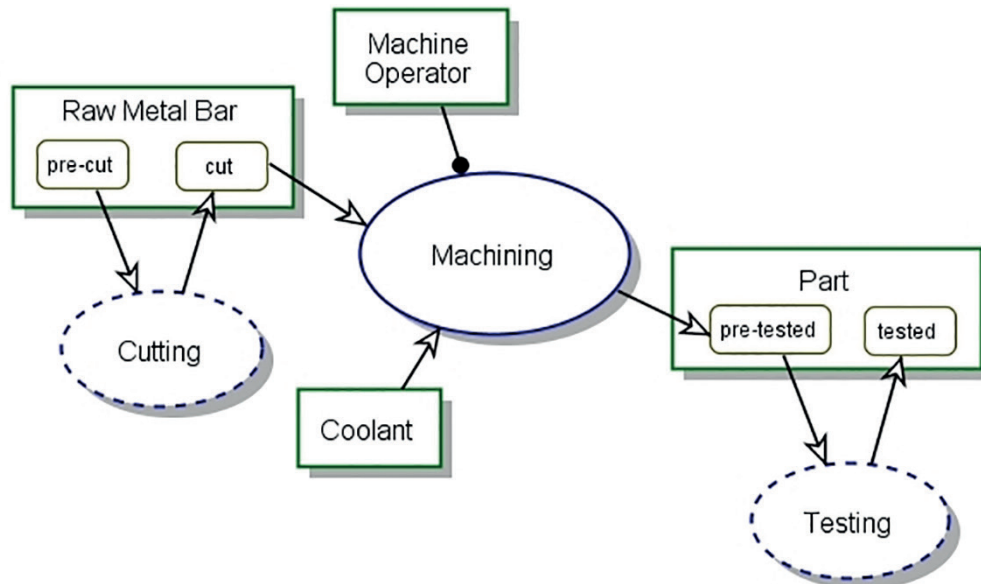
Graphically, a pair of arrows consisting of an arrow with a closed arrowhead from the input source state of the affectee to the affecting process, the input source link, and a similar arrow from that process to the output destination state of the affectee at process completion, the output destination 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** needs to 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 is environmental and physical.
Cutting changes **Raw Metal Bar** from **pre-cut** to **cut**.
Machining consumes **cut 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.

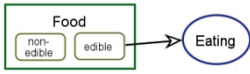
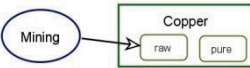
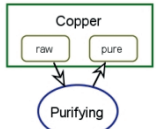
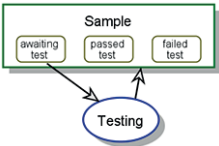
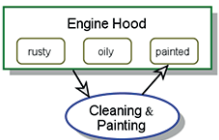
Graphically, a pair of arrows consisting of an arrow with a closed arrowhead from the affectee without specifying a particular state, the input link, and a similar arrow from that process to an output destination state of that affectee, the output link, shall denote the output-specified effect link.

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](#) summarizes the state-specified transforming links.

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.	consumer 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 <i>input link</i> and one state-specified <i>output link</i>)	The process changes the object from a specified input state via the <i>input link</i> to a specified output state via the <i>output link</i> .	 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 <i>input link</i> and one state-unspecified <i>output link</i>)	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 <i>input link</i> and one state-specified <i>output link</i>)	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** needs to be **sober** in order to qualify as an agent for the **Flying** process of an **Airplane**. In OPL: **Sober Pilot** handles **Flying**.

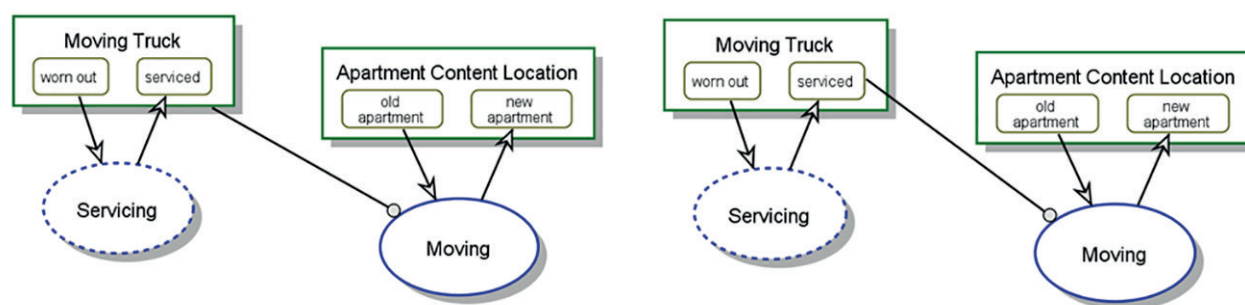
9.4.2 State-specified instrument link

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 **served** of **Moving Truck** is an instrument of **Moving**, meaning that if and only if **Moving Truck** is **served** can **Moving** take place.



Moving Truck is physical.
Moving Truck can be **worn out** or **served**.
Servicing is environmental and physical.
Servicing changes **Moving Truck** from **worn out** to **served**.
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**.

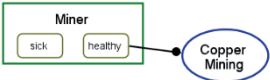
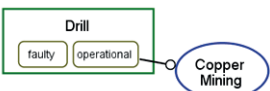
Moving Truck is physical.
Moving Truck can be **worn out** or **served**.
Servicing is environmental and physical.
Servicing changes **Moving Truck** from **worn out** to **served**.
Apartment Content Location is physical.
Apartment Content Location can be **old apartment** or **new apartment**.
Moving is physical.
Moving requires **served 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](#) summarizes the state-specified enabling links.

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 the operational semantics of OPM, 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 can 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 Since failure to complete successfully often results in undertime or overtime performance, exception links can serve other situations. In addition, all non-time related exceptions can be modelled using value ranges (see Clause C.6 for such usage).

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.

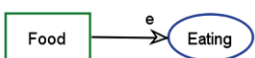
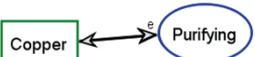
Graphically, a bidirectional arrow with closed arrowheads at each end between the object and the process with a small letter “e” annotation near the process end of the arrow, signifying event, shall denote the effect event link.

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](#) summarizes the transforming event links.

Table 5 — Transforming event links 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.

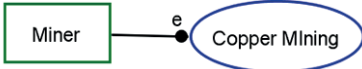
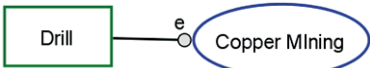
Graphically, a line with an empty circle resembling white lollipop at the terminal end extending from the instrument object to the process it initiates and enables with a small letter “e” annotation near the process end, signifying event, shall denote an instrument event link.

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

9.5.2.2.3 Enabling event links summary

[Table 6](#) summarizes the enabling event links.

Table 6 — Enabling event links summary

Name	Semantics	Sample OPD & OPL	Source	Destination
Agent event link	The agent—a human—both initiates and enables the process. The agent needs to 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 needs to 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.

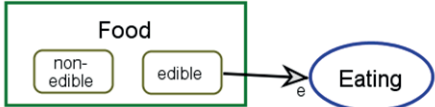
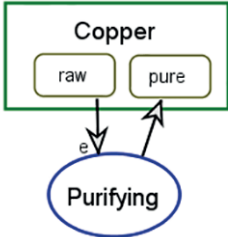
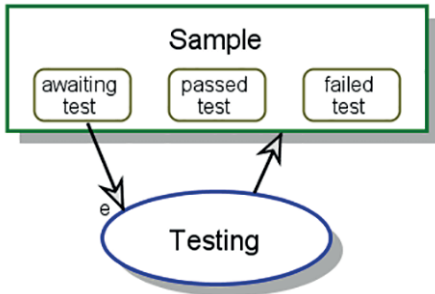
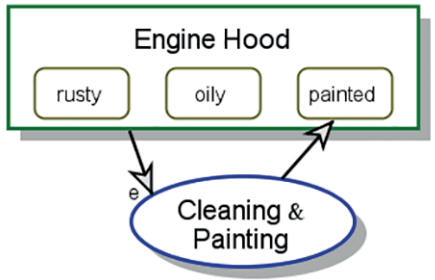
Graphically, the output-specified effect link with a small letter “e” annotation at the arrowhead end of the input link, signifying event, shall denote the output-specified effect event link.

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 links summary

[Table 7](#) summarizes the state-specified transforming event links.

Table 7 — State-specified transforming event links 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.	consumer 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 to passed test or failed 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.	 Rusty 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.

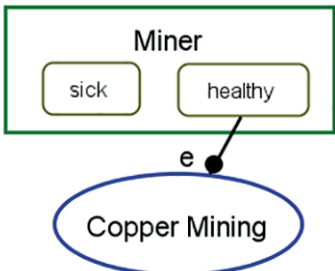
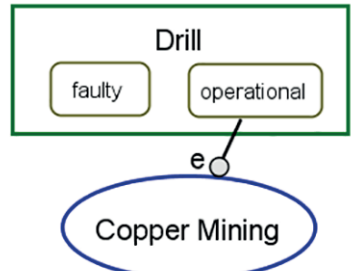
Graphically, the state-specified instrument link with a small letter “e” annotation near the process end of the link, signifying event, shall denote the state-specified instrument event link.

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 links summary

[Table 8](#) summarizes the state-specified enabling event links.

Table 8 — State-specified enabling event links 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 needs to 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 needs to 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. [Clause C.6](#) provides several ways to manage termination of a process because of a failure, especially [C.6.8](#).

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 can 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.

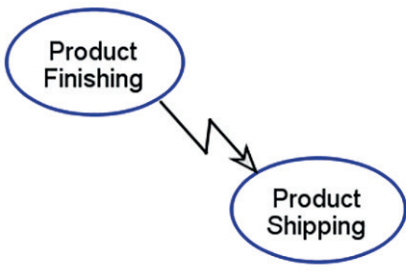
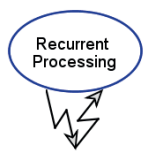
Graphically, a pair of invocation links, originating at the process and joining head to tail before ending back at the original process shall denote the self-invocation link.

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

9.5.2.5.3 Invocation links summary

[Table 9](#) summarizes the invocation links.

Table 9 — Invocation links 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**.

NOTE See [14.2.2.4.2](#) for additional detail regarding the semantics of "skip" and [Figure C.25](#) for several examples.

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.

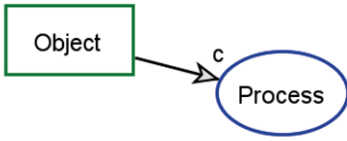
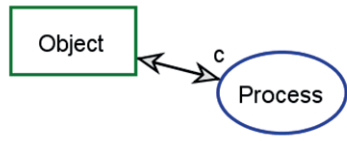
The syntax of the condition effect link OPL sentence shall be: **Process** occurs if **Object** exists, in which case **Process** affects **Object**, otherwise **Process** is skipped.

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 links summary

[Table 10](#) summarizes the condition transforming links.

Table 10 — Condition transforming links 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

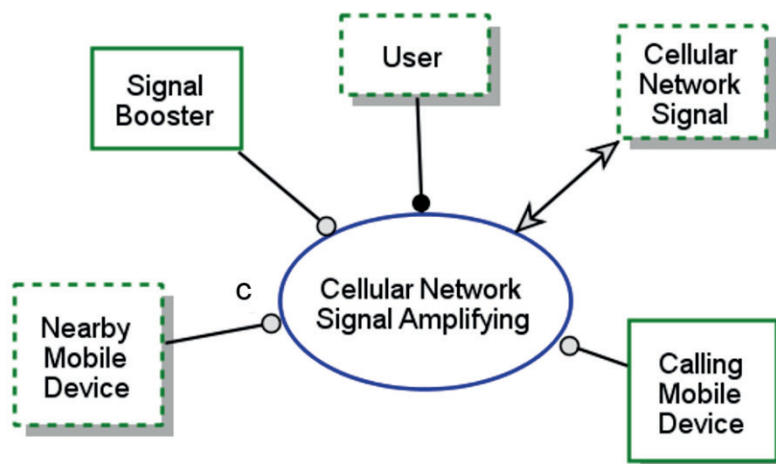
A condition instrument link shall be an annotated instrument link from an instrument to a process. If an instrument operational instance exists when an event initiates the process, then the presence of that instrument instance satisfies the process precondition with respect to that object. If evaluation 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 links summary

[Table 11](#) summarizes the basic condition enabling links.