INTERNATIONAL STANDARD

ISO/IEC 24744

First edition
2007-02-15

AMENDMENT 1
2010-02-01

Software Engineering — Metamodel for Development Methodologies

AMENDMENT 1: Notation

Ingénierie du logiciel — Métamodèle pour les méthodologies de développement

AMENDEMENT 1: Notation

Reference number
ISO/IEC 24744:2007/Amd.1:2010(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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

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

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

Amendment 1 to ISO/IEC 24744:2007 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 7, *Software and systems engineering*.

Software Engineering — Metamodel for Development Methodologies

AMENDMENT 1: Notation

Page iii, Contents

Add the following line after "**Annex B** (informative) **Mappings to Other Metamodelling Approaches**":

Annex C (informative) **Graphical Notation**

Page iv, Table of Figures

After the last entry ("Figure 9 – Support classes") add the following:

Figure C.1 – A lifecycle diagram showing the temporal structure of a complete method

Figure C.2 – A lifecycle diagram showing the content structure as well as the temporal structure of a method

Figure C.3 – An enactment diagram for the "Construction" phase kind of Figure C.2

Figure C.4 – A dependency diagram based on a refinement of Figure C.2

Figure C.5 – A process diagram showing the details of the "Requirements Engineering" and "Requirements Quality Assurance" processes

Figure C.6 – An action diagram showing how requirements-related task kinds interact with requirements-related work products

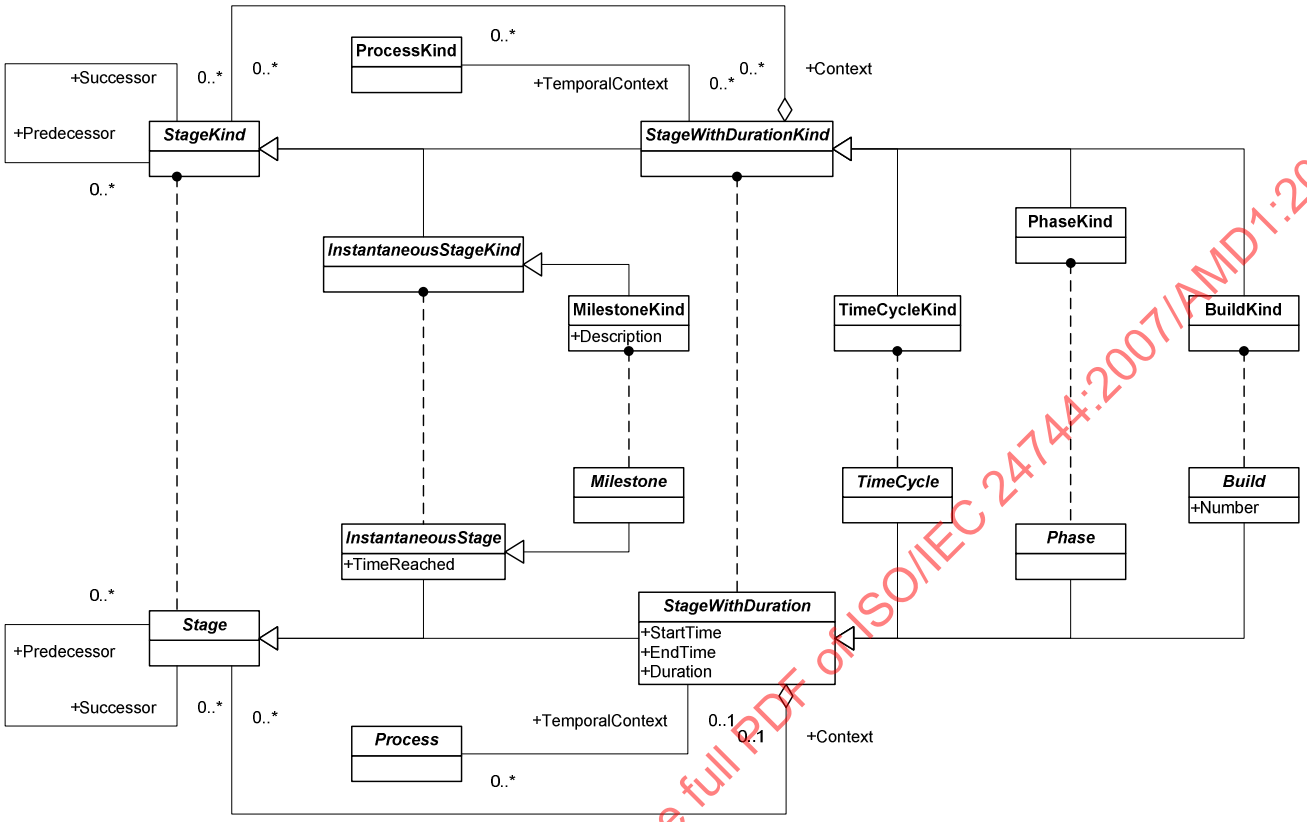
Page vi, Introduction

Add the following paragraph at the end of the Introduction:

This International Standard also presents a proposed notation for the ISO/IEC 24744 standard metamodel. The notation presented here is mainly graphical and supports most of the classes found in ISO/IEC 24744.

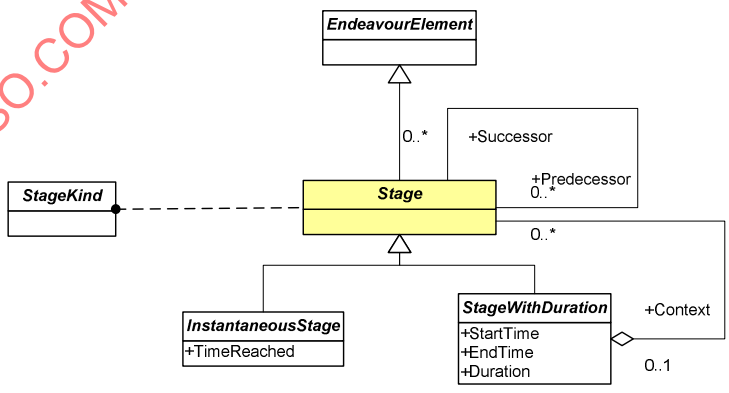
Page 11, Figure 5

Replace the figure with the following:



Page 46, 7.1.46

Replace the diagram with the following:



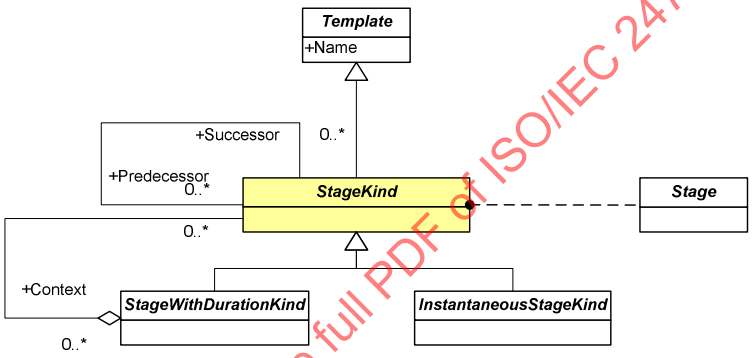
Page 47, 7.1.46.2

Add the following two new rows to the table:

Name	Role	To class	Semantics
OccursAfter	Successor	Stage	A successor stage occurs after some predecessor stages.
OccursBefore	Predecessor	Stage	A predecessor stage occurs before some successor stages.

Page 47, 7.1.47

Replace the diagram with the following:



Page 47, 7.1.47.2

Add the following two new rows to the table:

Name	Role	To class	Semantics
OccursAfter	Successor	StageKind	A successor stage kind occurs after some predecessor stage kinds.
OccursBefore	Predecessor	StageKind	A predecessor stage kind occurs before some successor stage kinds.

Page 78, before the Bibliography

Insert the following new annex:

Annex C
(informative)

Graphical Notation

C.1 Introduction

The metamodel of ISO/IEC 24744 contains classes that represent concepts from the method domain and classes that represent concepts from the endeavour domain. The notation presented here covers mainly the former, although some recommendations are given on how to represent the latter. Using this notation, methodologists or method engineers can represent method fragments and complete methodologies, and project managers can depict endeavours as they progress over time.

This notation has been designed to be easy to draw by hand as well as using a software tool on a computer. Special care has been taken in choosing symbols that convey the underlying concept, at least in most situations and to readers of most cultures and backgrounds. In addition, the symbols adopted by the notation exhibit visual resemblance (based on shapes and colours) to each other that mimic the structural relationships of the underlying concepts in the metamodel, establishing common “visual themes” for closely related concepts. Although colour is extensively used by this notation, since it helps identify symbols and symbol patterns with ease when displayed on a computer display or a colour printout, it is important to note that care has been taken to guarantee that greyscale and black and white versions of the same symbols are perfectly readable and identifiable. In this regard, colour does enhance diagram readability when it is available but, conversely, it can be avoided without too great a loss. Colour specification is done via RGB values in the sRGB (IEC 61966-2-1:1999) colour space.

C.1.1 Abstract Symbols

This notation introduces the concept of “abstract symbols”, i.e. symbols that depict instances of abstract classes. In principle, most notations only include symbols to depict instances of concrete classes, since abstract classes do not have direct instances. However, in some scenarios it is convenient to represent an entity in a diagram for which only the abstract type is known. This can be achieved by using so-called abstract symbols. For example, consider the case where a work product kind representing a certain system must be depicted in a diagram. A notation with only concrete symbols would force the diagram author to choose a specific concrete type of work product kind (such as document kind, model kind, software item kind etc.) in order to depict it. This notation includes an “abstract work product kind” symbol that allows the author to depict the above-mentioned system without specifying whether it is a model kind, a document kind, a software item kind etc. Abstract symbols usually consist of the simple shape from which all the concrete symbols in the visual theme are generated.

C.1.2 Notation Coverage

As a general principle, graphic symbols are given in the notation for every concrete (i.e. directly instantiable) class in the metamodel for which a graphical representation is considered to be appropriate. In addition, additional graphic symbols (abstract symbols) are given for every abstract class that is a direct ancestor (i.e. superclass) of said concrete classes. Abstract classes or higher ranks are considered to be too intangible as to be worth representing visually. For example, the DocumentKind class is concrete and suitable for visual representation, so a symbol for it is given in the notation; the WorkProductKind is abstract but a direct ancestor of DocumentKind, so an abstract symbol is given to it; on the contrary, the Template class, superclass of WorkProductKind, is too intangible and no graphical representation is given for it.

Specifically, the proposed notation for ISO/IEC 24744 covers the following classes:

— Stage-related classes, i.e. TimeCycleKind, PhaseKind, BuildKind and MilestoneKind as well as their direct ancestors StageWithDurationKind and InstantaneousStageKind.

— Work unit-related classes, i.e. ProcessKind, TaskKind and TechniqueKind as well as their direct ancestor WorkUnitKind. The related class Outcome is also covered.

— Work product-related classes, i.e. DocumentKind, ModelKind, SoftwareItemKind, HardwareItemKind and CompositeWorkProductKind, as well as their direct ancestor WorkProductKind.

— Producer-related classes, i.e. TeamKind, RoleKind and ToolKind, as well as their direct ancestor ProducerKind. The endeavour-level-only class Person is also covered.

— Constraint classes, i.e. PreCondition and PostCondition.

— "Relationship" classes, i.e. ActionKind, TaskTechniqueMappingKind and WorkPerformanceKind.

— Some support classes, i.e. Conglomerate and Guideline.

Wherever possible, these areas determine a set of "families" of symbols, which roughly correspond to branches in the specialization hierarchy in the metamodel. Within each such family, the shapes and colours of the icons for the subtypes are similar. In addition, notation is given for generic concepts, usually corresponding to relationships and links that may occur in a variety of situations.

The above bulleted list includes mostly method-domain classes. The corresponding endeavour-domain classes are also addressed by this notation. For the sake of coherence, the convention is adopted that an endeavour-domain class is always represented by the same symbol as used for its method-domain counterpart but using a dashed line. For example, the symbol for the MilestoneKind class is a small rotated blue square; this means that the symbol for the Milestone class is a similarly small rotated blue square in a dashed line. By using this convention, a type (method-domain) and its instances (endeavour-domain) are always represented by closely enough symbols but with a clear difference that makes them distinguishable. This convention is assumed throughout the remaining sections of this annex, and therefore explicit notation for endeavour-domain classes is not given.

The metamodel areas not covered by this notation include:

— Language-related classes, i.e. Language, Notation and ModelUnitKind. These classes work together to represent formal languages (plus their visual representations), which are better expressed by class diagrams, EBNF or other formalisms. A graphical representation is therefore considered not appropriate.

— Reference-related classes, i.e. Reference and Source. These classes are designed to allow method engineers to provide reference material about the "method chunks" (Section 5.1) that they create. Structured text is a better representation of this kind of information than graphical diagrams.

The following sections describe the notation in detail, including the symbols used, the syntax for their usage (derived from the metamodel) and the associated semantics (given by the mapping between the graphical symbols and the classes in the metamodel).

C.2 Notation Elements

C.2.1 Stages

A stage is a managed time frame within an endeavour (Section 7.1.46). Stages are partitioned into stage kinds by the StageKind class according to the abstraction level at which they work on the endeavour and the result that they aim to produce (Section 7.1.47).

Four concrete subtypes of StageKind are covered by this notation: TimeCycleKind, PhaseKind, BuildKind and MilestoneKind. The former three correspond to stages with duration and are, therefore, represented by broad symbols that can contain other elements. A rectilinear theme has been chosen to convey the idea of temporality. MilestoneKind, on the other hand, corresponds to instantaneous stages, and is consequently depicted by a narrower symbol that cannot contain nested elements. Colours for all these symbols belong to the blue-purple range. The name of the stage kind is shown inside the symbol.

C.2.1.1 StageWithDurationKind

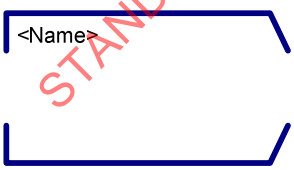
A stage with duration is a managed interval of time within an endeavour (Section 7.1.48). Stages with duration are partitioned into stage with duration kinds by the StageWithDurationKind class according to its abstraction level and the result it aims to produce (Section 7.1.49).

This is an abstract class, depicted by an abstract symbol, a horizontally oriented rectangle. This symbol tries to convey the idea of an empty container, inside which other elements can be shown. Line colour is navy blue (RGB 0, 0, 128) and fill colour is light blue-grey (RGB 225, 225, 255). The name of the stage with duration kind is shown inside the rectangle, in the top left corner, where the "<Name>" placeholder appears in the following figure.

C.2.1.2 TimeCycleKind

A time cycle is a managed interval of time within an endeavour for which the objective is the delivery of a final product or service (Section 7.1.59). Time cycles are partitioned into time cycle kinds by the TimeCycleKind class according to the type of outcomes that they aim to produce (Section 7.1.60).

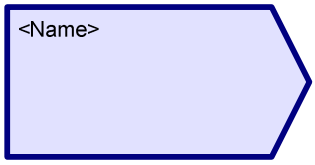
The symbol used to depict a time cycle kind is composed of two horizontal brackets with their right-hand side end bent outwards, simulating a truncated arrow head. These brackets delimit a rectangle within which symbols for other stage kinds can be shown. This symbol tries to convey the meaning that a time cycle kind comprises a collection of other stage kinds, hence the bracket analogy. Line colour is navy blue (RGB 0, 0, 128). The name of the time cycle kind is shown inside the symbol, in the top left corner, where the "<Name>" placeholder appears in the following figure.



C.2.1.3 PhaseKind

A phase is a managed interval of time within an endeavour for which the objective is the transition between cognitive frameworks (Section 7.1.31). Phases are partitioned into phase kinds by the PhaseKind class according to the abstraction level and formality of the result that they aim to produce (Section 7.1.32).

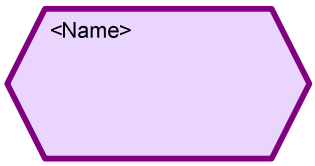
The symbol used to depict a phase kind is a pointed rectangle with the point facing to the right. Line colour is navy blue (RGB 0, 0, 128) and fill colour is light blue-grey (RGB 225, 225, 255). This symbol tries to convey the idea of temporality – hence the point, reminiscent of an arrow head. The name of the phase kind is shown inside the symbol, in the top left corner, where the "<Name>" placeholder appears in the following figure.



C.2.1.4 BuildKind

A build is a managed interval of time within an endeavour for which the major objective is the delivery of an incremented version of an already existing set of work products (Section 7.1.3). Builds are partitioned into build kinds by the BuildKind class according to the type of result that they aim to produce (Section 7.1.4).

The symbol used to depict a build kind is a double-pointed rectangle with the points facing to the right and left. This symbol tries to convey the idea of sequence, hence the double point, resembling dual arrow heads. Line colour is lilac (RGB 128, 0, 128) and fill colour is light purple (RGB 234, 213, 255). The name of the build kind is shown inside the symbol, on the top left corner, where the "<Name>" placeholder appears in the following figure.



C.2.1.5 InstantaneousStageKind

An instantaneous stage is a managed point in time within an endeavour (Section 7.1.16). Instantaneous stages are partitioned into instantaneous stage kinds by the InstantaneousStageKind class according to the kind of event that it signifies (Section 7.1.17).

This is an abstract class, depicted by an abstract symbol, a square. This symbol tries to convey the idea of a point in time, hence the similarity with other stage-related symbols (overall rectangular shape) but a smaller area. Since instantaneous stages are points in time rather than time spans, no other symbols can be shown inside this one. Line colour is bright blue (RGB 0, 0, 255) and fill colour is light blue (RGB 213, 213, 255). The name of the instantaneous stage kind is shown inside the square, centred, where the "<Name>" placeholder appears in the following figure.

C.2.1.6 MilestoneKind

A milestone is a managed point in time within an endeavour that marks some significant event in the endeavour (Section 7.1.20). Milestones are partitioned into milestone kinds by the MilestoneKind class according to their specific purpose and kind of event that they signify (Section 7.1.21).

The symbol used to depict a milestone kind is a small square rotated 45 degrees, resembling a diamond shape. This symbol tries to convey the idea of an event-marking point in time, hence the similarity with other stage-related symbols (points facing left and right) but a smaller area. It also resembles the symbol used by many project management software tools to depict milestones. Line colour is bright blue (RGB 0, 0, 255) and fill colour is light blue (RGB 213, 213, 255). The name of the milestone kind is shown inside the square, centred, where the “<Name>” placeholder appears in the following figure.

C.2.2 Work Units

A work unit is a job performed, or intended to be performed, within an endeavour (Section 7.1.67). Work units are partitioned into work unit kinds by the WorkUnitKind class according to their purpose within the endeavour (Section 7.1.68).

Three concrete subtypes of WorkUnitKind are covered by this notation: ProcessKind, TaskKind and TechniqueKind. None of these concepts are involved in whole/part relationships that may need nesting of symbols, so the symbols chosen to depict them are basic shapes and easily resizable to accommodate long names or abbreviations. All shapes are curvilinear. Colours for all these symbols belong to the green range. The name of the work unit kind is shown inside the symbol, centred. The minimum capability level (Section 7.1.68.1) of the work unit kind can be optionally shown inscribed in a small circle, inside the symbol.

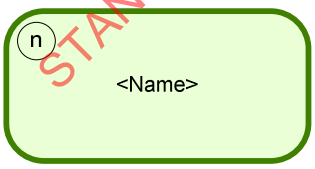
In addition, the Outcome class is also covered.

No abstract symbol is provided for WorkUnitKind because no abstract usage of work unit kinds is expected.

C.2.2.1 ProcessKind

A process is a large-grained work unit that operates within a given area of expertise within the endeavour (Section 7.1.35). Processes are partitioned into process kinds by the ProcessKind class according to the area of expertise in which they occur (Section 7.1.36).

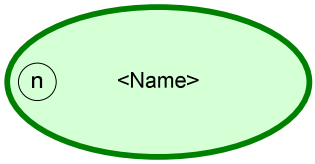
The symbol used to depict a process kind is a rounded rectangle or “roundangle”. Line colour is olive green (RGB 102, 153, 0) and fill colour is light olive green (RGB 234, 255, 213). The name of the process kind is shown inside the symbol, centred, where the “<Name>” placeholder appears in the following figure. The minimum capability level of the process kind is optionally shown in the top left corner inside the rectangle (“n” in the figure).



C.2.2.2 TaskKind

A task is a small-grained work unit that focuses on what must be done in order to achieve a given purpose within the endeavour (Section 7.1.50). Tasks are partitioned into task kinds by the TaskKind class according to their purpose within the endeavour (Section 7.1.51).

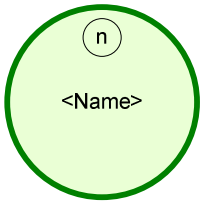
The symbol used to depict a task kind is an ellipse. Line colour is green (RGB 0, 128, 0) and fill colour is light green (RGB 213, 255, 213). The name of the task kind is shown inside the symbol, centred, where the "<Name>" placeholder appears in the following figure. The minimum capability level of the task kind is optionally shown towards the left edge inside the ellipse ("n" in the figure).



C.2.2.3 TechniqueKind

A technique is a small-grained work unit that focuses on how the given purpose may be achieved (Section 7.1.57). Techniques are partitioned into technique kinds by the TechniqueKind class according to their purpose within the endeavour (Section 7.1.58).

The symbol used to depict a technique kind is a circle. Line colour is green (RGB 0, 128, 0) and fill colour is light olive green (RGB 234, 255, 213). The name of the technique kind is shown inside the symbol, centred, where the "<Name>" placeholder appears in the following figure. The minimum capability level of the technique kind is optionally shown towards the upper edge inside the symbol ("n" in the figure).



C.2.2.4 Outcome

An outcome is an observable result of the successful performance of any work unit of a given kind (Section 7.1.29).

The symbol used to depict an outcome is a rectangle. Line colour is dark grey (RGB 51, 51, 51) and fill colour is light grey (RGB 221, 221, 221). The description of the outcome is shown inside the rectangle, flush left, where the "<Description>" placeholder appears in the following figure. The minimum capability level of the outcome is optionally shown towards the top left corner inside the rectangle ("n" in the figure). Outcome symbols are linked to the WorkUnitKind symbols to which they relate via generic links (Section C.2.8.3).

C.2.3 Work Products

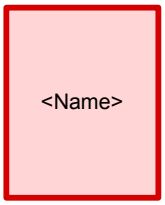
A work product is an artefact of interest for the endeavour (Section 7.1.65). Work products are partitioned into work product kinds by the WorkProductKind class according to the nature of their contents and the intention behind their usage (Section 7.1.66).

Five subtypes of WorkProductKind are covered by this notation: DocumentKind, ModelKind, SoftwareItemKind, HardwareItemKind and CompositeWorkProductKind. All of them are represented by vertically-oriented

symbols. Colours for all these symbols belong to the red-pink range. The name of the work product kind is shown inside the symbol, centred.

C.2.3.1 WorkProductKind

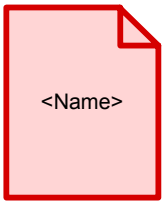
This is an abstract class, depicted by an abstract symbol, a vertically oriented rectangle. Line colour is red (RGB 213, 0, 0) and fill colour is light pink (RGB 255, 213, 213). The name of the work product kind is shown inside the rectangle, centred, where the “<Name>” placeholder appears in the following figure.



C.2.3.2 DocumentKind

A document is a durable depiction of a fragment of reality (Section 7.1.9). Documents are partitioned into document kinds by the DocumentKind class according to their structure, type of content and purpose (Section 7.1.10).

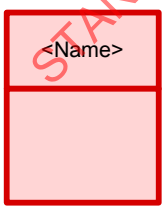
The symbol used to depict a document kind is a vertical rectangle with a dog-eared top right corner. This symbol depicts a sheet of paper. Line colour is red (RGB 213, 0, 0) and fill colour is light pink (RGB 255, 213, 213). The name of the document kind is shown inside the symbol, centred, where the “<Name>” placeholder appears in the following figure.



C.2.3.3 ModelKind

A model is an abstract representation of some subject that acts as the subject’s surrogate for some well-defined purpose (Section 7.1.22). Models are partitioned into model kinds by the ModelKind class according to their focus, purpose and level of abstraction (Section 7.1.23).

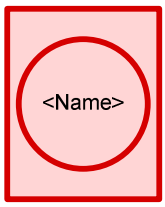
The symbol used to depict a model kind consists of a vertical rectangle divided in two compartments by a horizontal line closer to the top. This symbol resembles the symbol used by ISO/IEC 19501 to represent a class. Line colour is red (RGB 213, 0, 0) and fill colour is light pink (RGB 255, 213, 213). The name of the model kind is shown inside the symbol, centred, where the “<Name>” placeholder appears in the following figure.



C.2.3.4 SoftwareItemKind

A software item is a piece of software of interest to the endeavour (Section 7.1.43). Software items are partitioned into software item kinds by the SoftwareItemKind class according to their scope, requirements and features (Section 7.1.44).

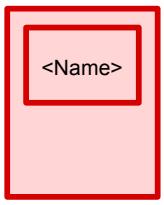
The symbol used to depict a software item kind is a vertical rectangle with an inscribed circle in the middle. This symbol depicts a DVD or CD inside its case and hence software. Line colour is red (RGB 213, 0, 0) and fill colour is light pink (RGB 255, 213, 213). The name of the software item kind is shown inside the symbol, centred, where the "<Name>" placeholder appears in the following figure.



C.2.3.5 HardwareItemKind

A hardware item is a piece of hardware of interest to the endeavour (Section 7.1.14). Hardware items are partitioned into hardware item kinds by the HardwareItemKind class according to their mechanical and electronic characteristics, requirements and features (Section 7.1.15).

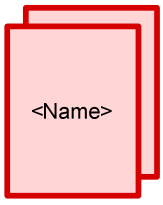
The symbol used to depict a hardware item kind is a vertical rectangle with a small rectangle nested in its upper half. This symbol depicts an integrated computer plus display assembly and hence hardware. Line colour is red (RGB 213, 0, 0) and fill colour is light pink (RGB 255, 213, 213). The name of the hardware item kind is shown inside the inner rectangle, centred across its top, where the "<Name>" placeholder appears in the following figure.



C.2.3.6 CompositeWorkProductKind

A composite work product is a work product composed of other work products (Section 7.1.5). Composite work products are partitioned into composite work product kinds by the CompositeWorkProductKind class according to their composition, i.e. the kinds of work products that are part of them (Section 7.1.6).

The symbol used to depict a composite work product kind is a pair of vertical rectangles "stacked" along the z-axis, simulating perspective. This symbol tries to convey the idea of composition, i.e. a work product made of multiple components. Line colour is red (RGB 213, 0, 0) and fill colour is light pink (RGB 255, 213, 213). The name of the composite work product kind is shown inside the symbol, centred, where the "<Name>" placeholder appears in the following figure.



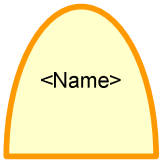
C.2.4 Producers

A producer is an agent that has the responsibility for executing work units (Section 7.1.37). Producers are partitioned into producer kinds by the ProducerKind class according to their area of expertise (Section 7.1.38).

Three subtypes of ProducerKind are covered by this notation: TeamKind, RoleKind and ToolKind. Also, the endeavour-domain class Person is covered. All of these classes are represented by symbols resembling half an ellipse as wide as tall. Colours for all these symbols belong to the orange-yellow range. The name of the producer kind is shown inside the symbol, centred.

C.2.4.1 ProducerKind

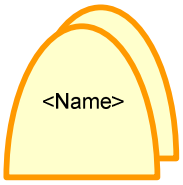
This is an abstract class, depicted by an abstract symbol. The symbol used to depict a work product kind is half an ellipse standing on its flat side. This symbol depicts a schematic human torso. Line colour is orange (RGB 255, 153, 0) and fill colour is light yellow (RGB 255, 255, 204). The name of the producer kind is shown inside the symbol, centred, where the “<Name>” placeholder appears in the following figure.



C.2.4.2 TeamKind

A team is an organized set of producers that collectively focus on common work units (Section 7.1.54). Teams are partitioned into team kinds by the TeamKind class according to their responsibilities (Section 7.1.55).

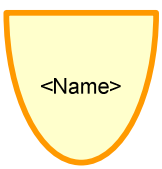
The symbol used to depict a team kind is a pair of half ellipses standing on their flat side and “stacked” along the z-axis, simulating perspective. This symbol depicts multiple human torsos, and hence the team. Line colour is orange (RGB 255, 153, 0) and fill colour is light yellow (RGB 255, 255, 204). The name of the team kind is shown inside the symbol, centred, where the “<Name>” placeholder appears in the following figure.



C.2.4.3 RoleKind

A role is a collection of responsibilities that a producer can take (Section 7.1.41). Roles are partitioned into role kinds by the RoleKind class according to the involved responsibilities (Section 7.1.42).

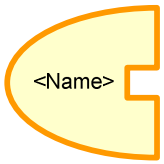
The symbol used to depict a role kind is half an ellipse standing on its round tip. This symbol depicts a face mask, and therefore a role that a producer may play. Line colour is orange (RGB 255, 153, 0) and fill colour is light yellow (RGB 255, 255, 204). The name of the role kind is shown inside the symbol, centred, where the “<Name>” placeholder appears in the following figure.



C.2.4.4 ToolKind

A tool is an instrument that helps another producer to execute its responsibilities in an automated way (Section 7.1.61). Tools are partitioned into tool kinds by the ToolKind class according to their features (Section 7.1.62).

The symbol used to depict a tool kind is a vertical half ellipse with its round tip pointing leftwards and a square indentation in the centre of its flat side. This symbol depicts the head of an open-end wrench or spanner, a prototypical tool. Line colour is orange (RGB 255, 153, 0) and fill colour is light yellow (RGB 255, 255, 204). The name of the tool kind is shown inside the symbol, centred, where the “<Name>” placeholder appears in the following figure.

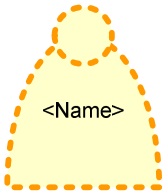


C.2.4.5 Person

A person is an individual human being involved in a development effort (Section 7.1.30).

The symbol used to depict a person is a vertical half ellipse standing on its flat side and a small circle superimposed on the ellipse tip. This symbol depicts a torso and head of a real person. Line colour is orange (RGB 255, 153, 0) and fill colour is light yellow (RGB 255, 255, 204). The name of the person is shown inside the symbol, centred, where the “<Name>” placeholder appears in the following figure.

Since the Person class belongs to the endeavour domain, line style is dashed.



C.2.5 Constraints

A constraint is a condition that holds or must hold at certain point in time (Section 7.1.8).

The two subtypes of Constraint are covered by this notation: PreCondition and PostCondition. Both of them are represented by long horizontal rectangular shapes. Colours for all these symbols belong to the grey range. Constraint symbols are linked to the ActionKind symbols they relate to via generic links (Section C.2.8.3).

No abstract symbol is provided for Constraint because no abstract usage of constraints is expected.

C.2.5.1 PreCondition

A precondition is a constraint that must be satisfied before an action of the associated kind can be performed (Section 7.1.34).

The symbol used to depict a precondition is a long horizontal rectangle with an arrow going out of a small square on the right-hand side. This symbol depicts a condition that must be met before control can flow into the associated action. Line colour is dark grey (RGB 51, 51, 51) and fill colour is light grey (RGB 221, 221, 221). The precondition expression appears inside the long rectangle, flush left, where the “<Expression>” placeholder appears in the following figure.

C.2.5.2 PostCondition

A postcondition is a constraint that is guaranteed to be satisfied after an action of the associated kind is performed (Section 7.1.33).

The symbol used to depict a postcondition is a long horizontal rectangle with an arrow going into a small square on the left-hand side. This symbol depicts a condition that is met once control flows out of the associated action. Line colour is dark grey (RGB 51, 51, 51) and fill colour is light grey (RGB 221, 221, 221). The postcondition expression appears inside the long rectangle, flush left, where the "<Expression>" placeholder appears in the following figure.

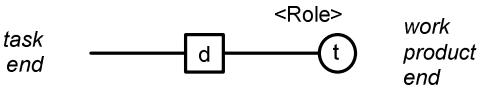
C.2.6 Relationship Concepts

Some concepts in the metamodel embody relationships between pairs of entities rather than denoting stand-alone entities themselves. These "relationship" concepts are represented in the notation by arcs between other symbols rather than by node-style symbols.

C.2.6.1 ActionKind

An action is a usage event performed by a task upon a work product (Section 7.1.1). Actions are partitioned into action kinds by the ActionKind class according to their cause (the specific task kind), their subject (the specific work product kind) and their type of usage (such as creation, modification etc.) (Section 7.1.2).

The symbol used to depict an action kind is an arc that goes from the symbol for the associated task kind to the symbol for the associated work product kind. The arc is a plain line with a small circle on the end of the work product kind. Line colour is black (RGB 0, 0, 0).



The type of usage is specified inside the small circle using an abbreviation ("t" in the figure), corresponding to the enumerators of enumerated type ActionType (Section 7.2.1). See Section C.4.1 for possible values. The role of the work product kind for this particular action kind, if any, can be shown close to the work product end. The optionality of the action kind can be shown using a deontic marker ("d" in the figure; see Section 0).

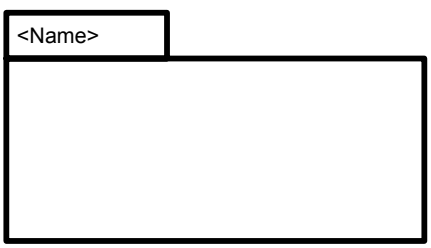
C.2.7 Support Concepts

The following support concepts from the metamodel are also covered by the notation.

C.2.7.1 Conglomerate

A conglomerate is a collection of related methodology elements that can be reused in different methodological contexts (Section 7.1.7). Conglomerates are not partitioned since they belong to the method rather than to the Endeavour domain.

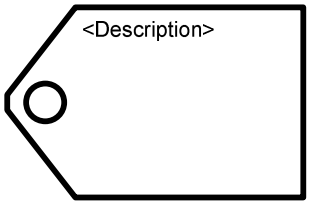
The symbol used to depict a conglomerate is a large rectangle, inside which other symbols can be nested, plus a smaller rectangle stacked on top of the top-left corner of the former. This symbol depicts a folder containing other elements. Line colour is black (RGB 0, 0, 0) and no fill colour is used. The name of the conglomerate is shown inside the small rectangle, flush left, where the "<Name>" placeholder appears in the following figure.



C.2.7.2 Guideline

A guideline is an indication of how a set of methodology elements can be used during enactment (Section 7.1.13). Guidelines are not partitioned since they belong to the method rather than to the endeavour domain.

The symbol used to depict a guideline is a rectangular shape with a pointed side to the left and a small circle next to this point, inside the shape. This symbol depicts a cardboard tag with a hole. Line colour is black (RGB 0, 0, 0) and no fill colour is used. The description of the guideline is shown inside the rectangle, flush left, where the "<Description>" placeholder appears in the following figure.



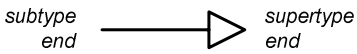
C.2.8 Links and Relationships

The notation incorporates some symbols for the various types of links and relationships that can appear in the diagrams.

C.2.8.1 Specialization Relationship

Conventional object-oriented specialization relationships are contemplated by this International Standard as a means to extend the metamodel and also as a mechanism to refine method-domain classes. Specialization is a relationship that occurs between types (rather than instances) and therefore it can only be applied to classes and class facets of clabjects.

The symbol used to depict a specialization relationship is an arc that goes from the symbol for the subtype to the symbol for the supertype. The arc is a plain line with a white triangular arrow head on the end of the supertype. Line colour is black (RGB 0, 0, 0).

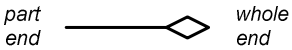


This symbol's appearance is identical to the specialization symbol used by ISO/IEC 19501.

C.2.8.2 Whole/Part Link

The metamodel contains a number of whole/part relationships, some of which, when instantiated, must use this symbol. It must be emphasized that this symbol represents links between instances (rather than relationships between types), and therefore can only be applied to objects and object facets of clabjects.

The symbol used to depict a whole/part link is an arc that goes from the symbol for the whole to the symbol for the part. The arc is a plain line with a white diamond-shaped arrow head on the end of the whole. Line colour is black (RGB 0, 0, 0).



This symbol is identical to the "white-diamond shared aggregation" symbol used by ISO/IEC 19501.

C.2.8.3 Generic Link

Any link between objects or object facets of clabjects that does not correspond to a whole/part relationship (but to a plain association) must be depicted using this symbol.

The symbol used to depict a generic link is an arc that goes between the symbols for the elements to be linked. The arc is a plain line with no adornments. Line colour is black (RGB 0, 0, 0).

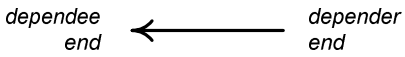
This symbol is identical to the basic association symbol used by ISO/IEC 19501.

C.2.8.4 Dependency Links

A dependency link shows that an entity abstractly depends on another entity. This relationship is the inverse of support; i.e. saying that entity A depends on entity B is equivalent to say that B supports A.

Dependency links are used when working at a high abstraction level and the specific type of dependency (or support) cannot be stated. For example, a dependency link can be used to express that a particular work product kind depends on a particular process kind without specifying which task kind in the process kind actually alters the work product kind, or how it does it.

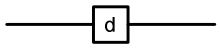
The symbol used to depict a dependency link is a directed arc that goes from the depender (or supported) into the dependee (or supporter). The arc is a plain line with a simple arrow head on the end of the depender. Line colour is black (RGB 0, 0, 0).



C.2.8.5 Deontic Marker

Deontic values are used by the metamodel to indicate optionality or degrees of recommendation in various situations. Deontic relationships are often depicted by this notation as arcs. A deontic marker can be embedded in an arc (such as a generic link or a support link) when necessary.

The symbol used to depict a deontic value in an arc is a small square embedded in the arc containing an abbreviation of the appropriate deontic value ("d" in the figure), corresponding to the enumerators of enumerated type DeonticValue (Section 7.2.2). See Section 0 for possible values. Line colour is black (RGB 0, 0, 0).



If the deontic value is Mandatory, the deontic marker can be omitted. In other words, Mandatory is the default deontic value assumed when no value is explicitly expressed.

C.2.8.6 Topological Containment

The metamodel contains a number of whole/part relationships, some of which, when instantiated, must be represented by topological containment of symbols rather than by any symbol in particular. Topological containment of symbols represents links between instances (rather than relationships between types). It can therefore only be applied to objects and object facets of clabjects.

Topological containment is made possible by showing the symbols for the part elements within the visual boundaries of the symbol for the whole element.

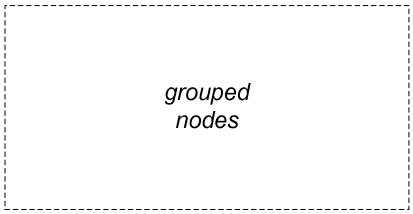
C.2.9 Diagramming Support

The following symbols are designed to support in laying out diagrams and organising diagram element on the page. They do not possess metamodel-related semantics.

C.2.9.1 Node Grouping

Sometimes, multiple nodes in a diagram show similar arcs connecting them to other nodes, resulting in confusion and low readability due to the overabundance of arcs. In these cases, the nodes in question can be grouped into a node group and the arcs drawn only once, which reduces confusion and enhances readability.

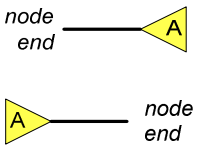
The symbol used to depict a node group is a dashed rectangle with no filling. Line colour is black (RGB 0, 0, 0). The rectangle must surround the nodes that are meant to be grouped; any arcs connected to the rectangle are interpreted as arcs connected to each of the contained nodes.



C.2.9.2 Callouts

Sometimes, arcs in a diagram can get very convoluted and hard to follow. In these cases, an arc can be broken into short segments attached to the respective nodes, and a callout symbol attached to each of these segments to “virtually” link them together. A callout label (usually a single letter or number) is placed inside the callout symbol so that both halves of the arc are easily matched by the reader when scanning a page.

The symbol to depict a callout is a small isosceles triangle pointing into the interrupted arc. The callout label (“A” in the figure) is shown inside the triangle. Line colour is black (RGB 0, 0, 0) and fill colour is bright yellow (RGB 255, 255, 79).



C.3 Diagram Types

C.3.1 Introduction

Several types of diagrams can be created using this notation. In order of decreasing level of abstraction, they are:

— Lifecycle diagrams, which represent the overall structure of a method (or part of it).

— Enactment diagrams, which represent a specific endeavour (or part of it) and its relationship to the corresponding method.

— Dependency diagrams, which represent the abstract support/dependency relationships among the major components (i.e. producer kinds, work unit kinds and work product kinds) of a methodology.

— Process diagrams, which describe the details of the process kinds used in a method.

— Action diagrams, which show the detailed usage interactions between task kinds and work product kinds.

As in other engineering disciplines, a collection of different diagrams can be used to show different views of the same underlying method or endeavour. Therefore, no single diagram should be expected to depict every single detail of the underlying method or endeavour.

The following sections describe each of the above listed diagram types in detail.

C.3.2 Lifecycle Diagrams

Lifecycle diagrams represent the overall structure of a method or a portion of a method. This structure has two aspects:

— A temporal aspect, corresponding to the configuration of stage kinds to compose a complete method.

— A content aspect, corresponding to the process kinds that may be executed within each stage kind.

The temporal aspect can be informally described as the "*when*", whereas the content aspect can be described as the "*what*". A lifecycle diagram always shows the temporal aspect of a method; the content aspect can be optionally shown as well.

C.3.2.1 Elements

The elements that appear inside a lifecycle diagram are stage kinds and process kinds. Topological containment is extensively used to depict temporal framing, such as a process being performed within a phase or a phase occurring within a time cycle.

Generic links are used to join stage kinds inside a common container. These links depict the OccursAfter and OccursBefore relationships between stage-related classes in the metamodel.

C.3.2.2 Examples

Consider the example shown in Figure C.1. A time cycle kind named "OPEN/Metis Project" is introduced, which contains a sequence of phase kinds: "Determination of Needs", "Definition", "Construction", etc. Some milestone kinds mark transitions between phase kinds, such as "M0". Some phase kinds are iterative, shown by nested build kinds such as "Construction Build". Build kinds, in turn, may contain nested stages as well.

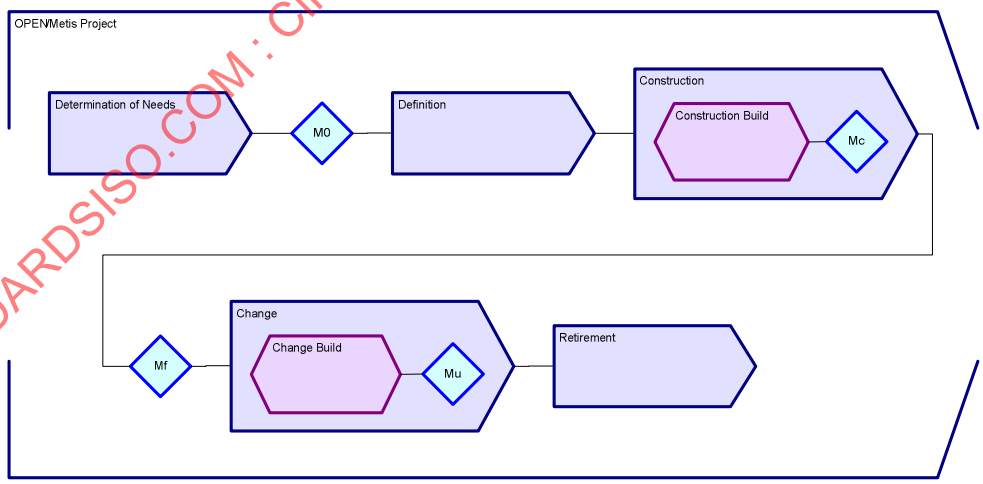


Figure C.1 — A lifecycle diagram showing the temporal structure of a complete method.

Figure C.1 depicts the temporal structure of a method. As we have said, it is possible to show the content structure as well. This is accomplished by including process kind symbols inside the appropriate stage kind symbols. Figure C.2 shows an example.

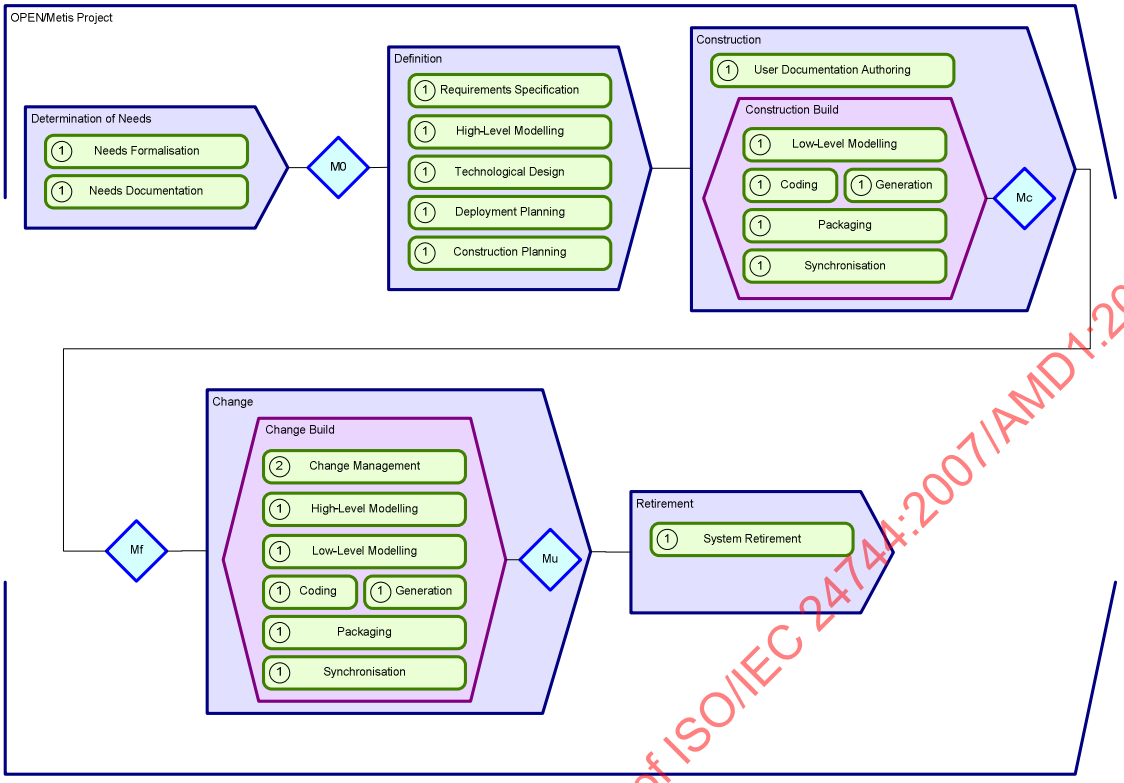


Figure C.2 — A lifecycle diagram showing the content structure as well as the temporal structure of a method.

Notice how stage kinds can contain both nested stage kinds (such as “Construction Build” inside “Construction”) and process kinds (“User Documentation Authoring”).

C.3.3 Enactment Diagrams

Enactment diagrams are a variation of lifecycle diagrams that include a Gantt chart linked to the symbols of the lifecycle diagram. Thus, enactment diagrams show a specific enactment of a particular method or method subset. For each stage kind and process kind in the lifecycle diagram, zero or more bars in the Gantt chart may appear, representing endeavour-domain occurrences of the method-domain specification.

C.3.3.1 Elements

The elements that appear inside an enactment diagram are the same that appear inside a lifecycle diagram, plus a Gantt chart. This notation does not specify any particular representation for Gantt charts.

C.3.3.2 Examples

Consider the “Construction” phase kind in Figure C.2, which contains a process kind plus a collection of process kinds and a milestone kind within a build kind. Figure C.3 shows a sample enactment diagram for this phase kind.

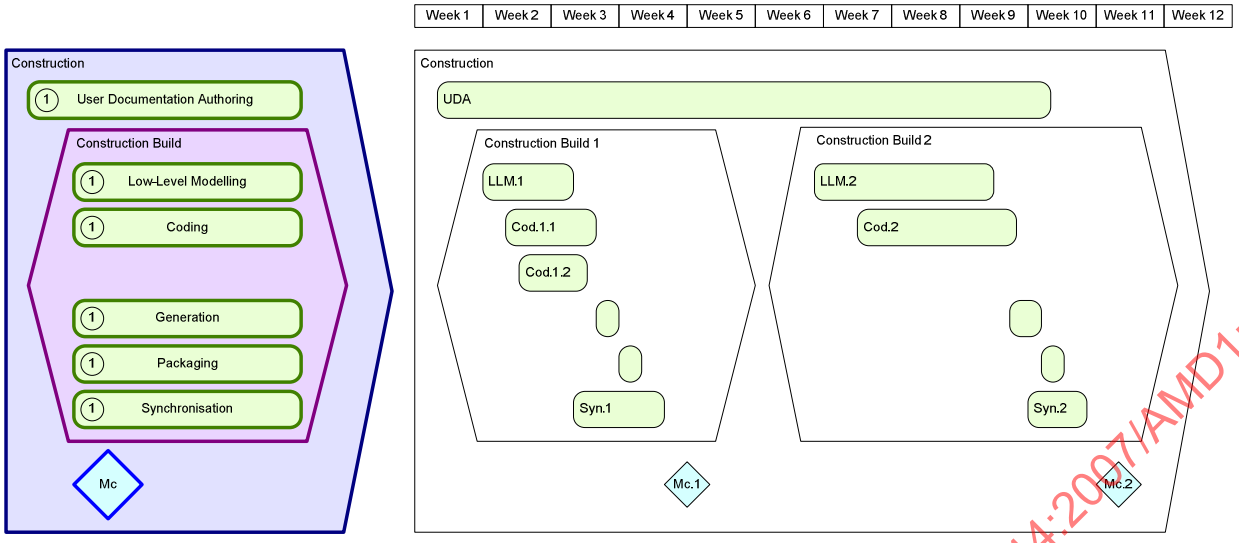


Figure C.3 — An enactment diagram for the "Construction" phase kind of Figure C.2.

The left-hand side of the enactment diagram represents the method-domain specification and is virtually identical to a lifecycle diagram, the only difference being that, in this case, symbols are laid out in a strict vertical fashion in order to leave the horizontal axis to represent time. A timeline can be optionally shown running along the horizontal axis if necessary. The right-hand side of the enactment diagram contains a Gantt chart in which bars corresponding to processes in the method are vertically aligned with the corresponding symbol in the lifecycle diagram to its left. For example, the bar labelled "UDA" in Figure C.3 represents the enactment of the "User Document Authoring" process kind in the method. Similarly, bars labelled "LLM.1" and "LLM.2" in the Gantt chart represent enactments of the "Low-Level Modelling" process kind in the method. Notice that bars labelled "Cod.1.1" and "Cod.1.2" represent time-overlapping enactments of the "Coding" process kind, and therefore a gap must be left in the left-hand side of the diagram for them to fit. Enactments of milestone kinds can be shown in a similar fashion. Also, the time span covered by stage kinds with duration (such as phase kinds or build kinds) are shown in the Gantt chart by surrounding the appropriate process bars with a boundary that reproduces the appearance of the associated method element. From a formal perspective, it can be said that the symbols in the Gantt chart represent instances of the elements depicted in the lifecycle diagram. The number of instances for each method element will depend on the semantics of the element; for example, Figure C.3 shows two instances of the "Construction Build" build kind.

It is worth noting that the bars, symbols and boundaries in the Gantt chart (i.e. the right-hand side of the enactment diagram) correspond to endeavour-domain entities. For example, the bar labelled "UDA" represents an instance of the Process class in the metamodel, and the boundaries labelled "Construction Build 1" and "Construction Build 2" represent instances of the Build class in the metamodel. However, Gantt-chart notation rather than endeavour-domain notation (Section C.1.2) is used to maintain the look and feel of a Gantt chart.

It is also important to stress that enactment diagrams are meant to provide an intuitive way to depict a high-level view of subsets of a methodology as being enacted over time, and are not supposed to serve as an enactment-management tool capable of showing implementation details.

C.3.4 Dependency Diagrams

Dependency diagrams represent the abstract dependency/support relationships between the major elements of a methodology, involving producer kinds, work unit kinds and work product kinds.

Dependency diagrams are useful to depict abstract relationships between method elements, without the need to determine how these dependencies are to be implemented. This is especially useful during method development. Dependency diagrams can be fleshed out into process diagrams and action diagrams when more detail is needed.