

Australian Standard™

Standard metamodel for software development methodologies



This Australian Standard was prepared by Committee IT-015, Software and Systems Engineering. It was approved on behalf of the Council of Standards Australia on 31 May 2004.

This Standard was published on 23 August 2004.

The following are represented on Committee IT-015:

Australian Computer Society
Australian Information Industry Association
Australian Society for Technical Communication (NSW)
Australian Software Metrics Association
Griffith University
New Zealand Organization for Quality
Software Quality Association (ACT)
Software Quality Association (NSW)
Sydney SPIN Group (Software Process Improvement Network)
Systems Engineering Society of Australia
The University of New South Wales
University of South Australia
University of Technology, Sydney

Keeping Standards up-to-date

Standards are living documents which reflect progress in science, technology and systems. To maintain their currency, all Standards are periodically reviewed, and new editions are published. Between editions, amendments may be issued. Standards may also be withdrawn. It is important that readers assure themselves they are using a current Standard, which should include any amendments which may have been published since the Standard was purchased.

Detailed information about Standards can be found by visiting the Standards Web Shop at www.standards.com.au and looking up the relevant Standard in the on-line catalogue.

Alternatively, the printed Catalogue provides information current at 1 January each year, and the monthly magazine, *The Global Standard*, has a full listing of revisions and amendments published each month.

Australian Standards™ and other products and services developed by Standards Australia are published and distributed under contract by SAI Global, which operates the Standards Web Shop.

We also welcome suggestions for improvement in our Standards, and especially encourage readers to notify us immediately of any apparent inaccuracies or ambiguities. Contact us via email at mail@standards.org.au, or write to the Chief Executive, Standards Australia International Ltd, GPO Box 5420, Sydney, NSW 2001.

This Standard was issued in draft form for comment as DR 04154.

Australian Standard™

Standard metamodel for software development methodologies

First published as AS 4651—2004.

COPYRIGHT

© Standards Australia International

All rights are reserved. No part of this work may be reproduced or copied in any form or by any means, electronic or mechanical, including photocopying, without the written permission of the publisher.

Published by Standards Australia International Ltd GPO Box 5420, Sydney, NSW 2001, Australia

ISBN 0 7337 6195 X

PREFACE

This Standard was prepared by the Australian members of the Joint Standards Australia/Standards New Zealand Committee IT-015, Software and Systems Engineering. After consultation with stakeholders in both countries, Standards Australia and Standards New Zealand decided to develop this Standard as an Australian Standard rather than an Australian/New Zealand Standard.

The objective of this Standard is to define a metamodel that establishes a formal framework for the definition and extension of software development methodologies, including their major three aspects: the process to follow, the products to use and generate, and the people involved. This metamodel can serve as a formal basis for the definition and extension of any software development methodology, and will be typically used by methodologists, method engineers or process engineers while undertaking such tasks.

The terms ‘normative’ and ‘informative’ have been used in this Standard to define the application of the appendix to which they apply. A ‘normative’ appendix is an integral part of a Standard, whereas an ‘informative’ appendix is only for information and guidance.

CONTENTS

	<i>Page</i>
INTRODUCTION	4
 SECTION 1 SCOPE AND GENERAL	
1.1 SCOPE	5
1.2 APPLICATION	5
1.3 CONFORMANCE	5
1.4 NORMATIVE REFERENCES	5
1.5 TERMS AND DEFINITIONS	5
 SECTION 2 INTRODUCTION TO THE SMSDM	
2.1 HIGHLY ABSTRACT VIEW	8
2.2 ABSTRACT VIEW AND CORE CLASSES	8
2.3 PROCESS CLASSES	9
2.4 PRODUCER CLASSES	10
2.5 PRODUCT CLASSES	11
2.6 CONNECTION BETWEEN PROCESS AND PRODUCT	12
2.7 SUPPORT CLASSES	13
 SECTION 3 ELEMENTS OF SMSDM	
3.1 CLASSES IN THE SMSDM	15
3.2 ENUMERATORS IN THE SMSDM	49
 SECTION 4 USAGE OF SMSDM	
4.1 USAGE RULES AND GUIDELINES	50
 APPENDICES	
A PHILOSOPHY	52
B WORKED EXAMPLE	56
C MAPPINGS TO OTHER METAMODELLING APPROACHES	62
D BIBLIOGRAPHY	67

INTRODUCTION

Software development methodologies may be described in the context of an underpinning metamodel, but the precise mechanisms that permit them to be defined in terms of their metamodels are usually difficult to explain and do not cover all needs. For example, it is difficult to devise a practice that allows the definition of properties of the elements that compose the methodology and, at the same time, of the entities (such as work products) created when the methodology is applied. This Standard introduces the Standard Metamodel for Software Development Methodologies (SMSDM henceforth), a comprehensive metamodel that makes use of a new approach to defining methodologies based on the concept of powertype. The SMSDM combines key advantages of other metamodelling approaches with none of their drawbacks, allowing the seamless integration of process, modelling and documentational aspects of methodologies.

Various methodologies are defined, used, or implied by a growing number of standards and it is desirable that the concepts used by each methodology be harmonised. The vehicle for harmonization is SMSDM. Conformance to this metamodel will ensure a consistent approach to defining each methodology with consistent concepts and terminology.

STANDARDS AUSTRALIA

Australian Standard

Standard metamodel for software development methodologies

SECTION 1 SCOPE AND GENERAL

1.1 SCOPE

This Standard defines a metamodel that establishes a formal framework for the definition and extension of software development methodologies, including their major three aspects: the process to follow, the products to use and generate, and the people involved.

1.2 APPLICATION

This metamodel can serve as a formal basis for the definition and extension of any software development methodology, and will be typically used by methodologists, method engineers or process engineers while undertaking such tasks.

1.3 CONFORMANCE

A software development methodology is conformant to this Standard if it has been generated from the metamodel defined in Section 3 Elements of SMSDM by using the rules described in Section 4.

1.4 NORMATIVE REFERENCES

This Standard does not call up any normative references.

1.5 TERMS AND DEFINITIONS

1.5.1 Terms

The following terms are used throughout this Standard.

1.5.1.1 *Methodology*

Specification of the process to follow and the work products to be used and generated, plus consideration of the people and tools involved, during a software development effort. A methodology specifies the process to be executed, usually as a set of related activities, tasks and/or techniques, together with what work products must be manipulated at each moment, possibly including models, documents and other outputs. In turn, specifying the models that must be developed implies defining the basic building blocks that should be used.

1.5.1.2 *Metamodel*

Specification of the concepts, rules and relationships used to define a methodology.

1.5.1.3 *Project*

Software development effort aimed at the construction of a software system through the application of a methodology. Sometimes, the terms ‘project-level’ or ‘project layer’ are used to reference the conceptual entities that exist at the level of projects, as opposed to the entities that exist at the level of a method or a metamodel.

1.5.1.4 *Software system*

Final product of a project, consisting of computer programs, procedures, and associated documentation and data (based on the definition in [10]).

1.5.1.5 *Methodology element*

Simple component of a methodology. Usually, methodology elements include the specification of what tasks, activities, techniques, models, documents, languages and/or notations can or must be used when applying the methodology. Methodology elements are related to each other, comprising a network of abstract concepts that the project manager will have to deal with. Typical methodology elements are Capture Requirements, Write Code for Methods (kinds of tasks), Requirements Engineering, High-Level Modelling (kinds of activities), Pseudo-code, Dependency Graphs (notations), Class, Attribute (kinds of model building blocks), Class Model, Class Diagram, Requirements Specification (kind of work products).

1.5.1.6 *Project element*

Simple component of a project. During the execution of a project, the development team creates a number of project entities, such as classes, attributes, models, documents, and tasks. Some examples of project entities are Customer, Invoice (classes), Name, Age (attributes), High-Level Class Model number 17 (a model), System Requirements Description (a document), Coding Cycle number 2, Coding Cycle number 3 (tasks).

1.5.1.7 *Generation*

Act of defining and, optionally, describing a methodology in the context of a particular metamodel. Generating a methodology includes explaining the structural position and semantics of each methodology element using the selected metamodel. Thus, what methodology elements are possible, and how they relate to each other, are constrained by such a metamodel. Usually, methodologists (or method/process engineers) perform generation, yielding a complete and usable methodology for project managers.

1.5.1.8 *Enactment*

Act of applying a methodology to a particular project. Enacting a methodology includes using the existing generated methodology to create project entities and, eventually, obtain the targeted software system. Thus, what kinds of project elements can be created, and how they relate to each other, is governed by the methodology being used. Usually, project managers, together with the rest of the development team, perform enactment.

1.5.2 Instruments defining SMSDM

The SMSDM is defined using different kinds of instruments that complement each other. These are:

(a) **Definitions**

Each concept in the SMSDM is defined using natural language. Also, a description is given, including the context in which the concept occurs and its most distinctive properties.

(b) **Class diagrams**

Concepts in the SMSDM are formalized as classes, and class diagrams are used to show their attributes and relationships. UML 1.4 is used throughout with some noticeable exceptions. First, a special notation is used to depict powertype patterns, consisting of a dashed line between the powertype and the partitioned type with a round end on the side of the powertype. Secondly, 'white diamonds' are used to depict whole/part relationships without making any reference to their secondary characteristics (see [8] for more details).

(c) **Text tables**

Text tables are included to provide additional descriptions of attributes and relationships.

(d) **Mappings to other approaches**

Each concept in the SMSDM is related to equivalent or similar concepts in other metamodeling approaches, so that the transition is easier.

These mechanisms are used simultaneously.

The philosophy of the SMSDM is to offer broad coverage for all the issues often found in methodology definition avoiding, at the same time, unnecessary structural constraints on the resultant methodologies. Therefore, only a minimal set of attributes and associations is provided by the metamodel. Using powertype pattern instantiation (see Clause 4.1.2.4) and thanks to the usage of powertypes in the metamodel, additional attributes and associations can be easily added to the methodology layer.

NOTE: Appendix A describes the general philosophy underpinning the SMSDM.

SECTION 2 INTRODUCTION TO THE SMSDM

2.1 HIGHLY ABSTRACT VIEW

From the most abstract perspective, the SMSDM defines the classes *MethodologyElement* and *ProjectElement* to represent, respectively, elements in the methodology and the project layers. *MethodologyElement*, in turn, is specialized into *Resource* and *Template*, corresponding to methodology elements that are used ‘as is’ at the project level (i.e. resources) and methodology elements that are used by instantiation at the project level (i.e. templates). Since *Template* is the abstract type of all elements at the methodology level, and *ProjectElement* is the abstract superclass of the same elements, these two classes form a powertype pattern in which *Template* is the powertype, *ProjectElement* is the partitioned type and *Template.Name* is the discriminant. See Figure 1 for a graphical representation.

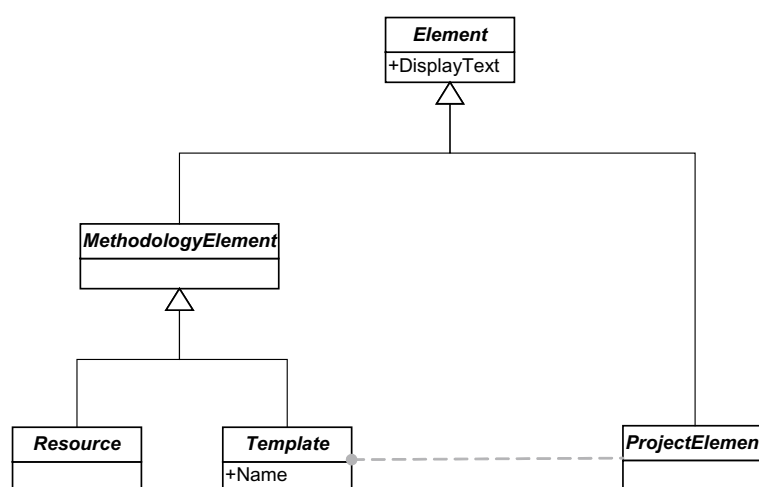


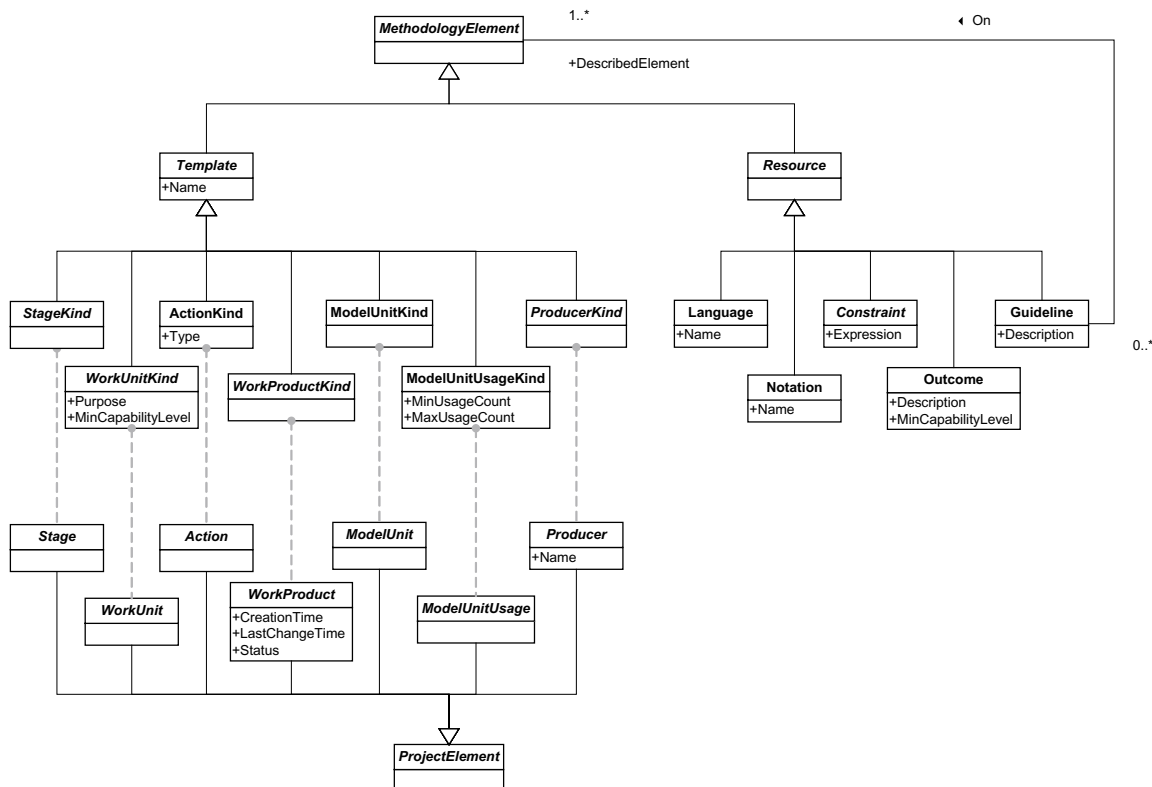
FIGURE 1 HIGHLY ABSTRACT VIEW OF THE SMSDM

At the same time, a top class *Element* is defined to generalize *MethodologyElement* and *ProjectElement* and allow homogeneous treatment of all elements across the methodology and project layers when necessary. The *DisplayText* attribute of *Element* gives a name or description of each instance suitable to be shown to final users.

2.2 ABSTRACT VIEW AND CORE CLASSES

There are three clusters of core classes: methodology templates, specializing from *Template*; methodology resources, specializing from *Resource*, and project classes, specializing from *ProjectElement*.

The powertype pattern formed by *Template* and *ProjectElement* is refined into more specialized powertype patterns formed by subclasses of these two, namely *StageKind* and *Stage* (representing managed intervals of time, or points in time, within a project, i.e. the temporal aspect of the process), *WorkUnitKind* and *WorkUnit* (functionally cohesive operations performed during a project, i.e. the job aspect of the process), *WorkProductKind* and *WorkProduct* (any artefacts of value developed or used during a project), *ActionKind* and *Action* (representing the specific act or usage of a given work product by some types of work units), *ProducerKind* and *Producer* (final responsible agents for actions, usually human), *ModelUnitKind* and *ModelUnit* (basic building blocks to be used in creating work products) and *ModelUnitUsageKind* and *ModelUnitUsage* (representing specific usages of a given model unit in a given work product). See Figure 2 for a graphical depiction.



NOTE: The powertype pattern relationship between template and project element is not shown for clarity.

FIGURE 2 ABSTRACT VIEW OF SMSDM, SHOWING THE CORE CLASSES IN THE METAMODEL

At the same time, Resource is specialized into Language (a set of interrelated model unit kinds that can be used to construct certain model kinds), Notation (a set of perceptible artefacts, usually graphical, which can be used to depict specific work products), Guideline (rules and directives about the appropriate usage of a given methodology element), Constraint (a condition related to the execution of an action) and Outcome (a visible result of the performance of a given work unit).

2.3 PROCESS CLASSES

The WorkUnit/*Kind powertype pattern is specialized into Workflow/*Kind (a cohesive yet heterogeneous collection of tasks that achieves a set of related goals), Task/*Kind (the smallest manageable unit of work, described in a declarative way) and Technique/*Kind (specific means to achieve a goal, described in a prescriptive way). In turn, Workflow/*Kind is further specialized into Activity/*Kind (an ongoing responsibility related to some area of expertise) and Process/*Kind (a large-grained unit of work that is atomically executable).

WorkUnitKind is characterized by a purpose and a minimum capability level at which it can be performed, and is related to Outcome in a one-to-many fashion, so a set of outcomes can be defined for each specific kind of work unit. Also, WorkUnit/*Kind holds a whole/part relationship to Task/*Kind, so any work unit or work unit kind can be defined as a collection of tasks or task kinds, respectively. This allows for the recursive definition of units of work down to the necessary level of detail.

© Standards Australia

Document details: AS 4651-2004



Producer/*Kind is specialized into Role/*Kind (a specific set of responsibilities, often associated to a given area of expertise, that can be taken by a producer), Tool/*Kind (an automated tool, usually in the form of software) and Team/*Kind (a set of other producers). Producer has an additional subclass, Individual, which allows taking into account individual persons at the project level. Producer/*Kind is also related to WorkUnit/*Kind, so links between units of work and the assigned producers are possible. See Figure 4 for a graphical depiction.

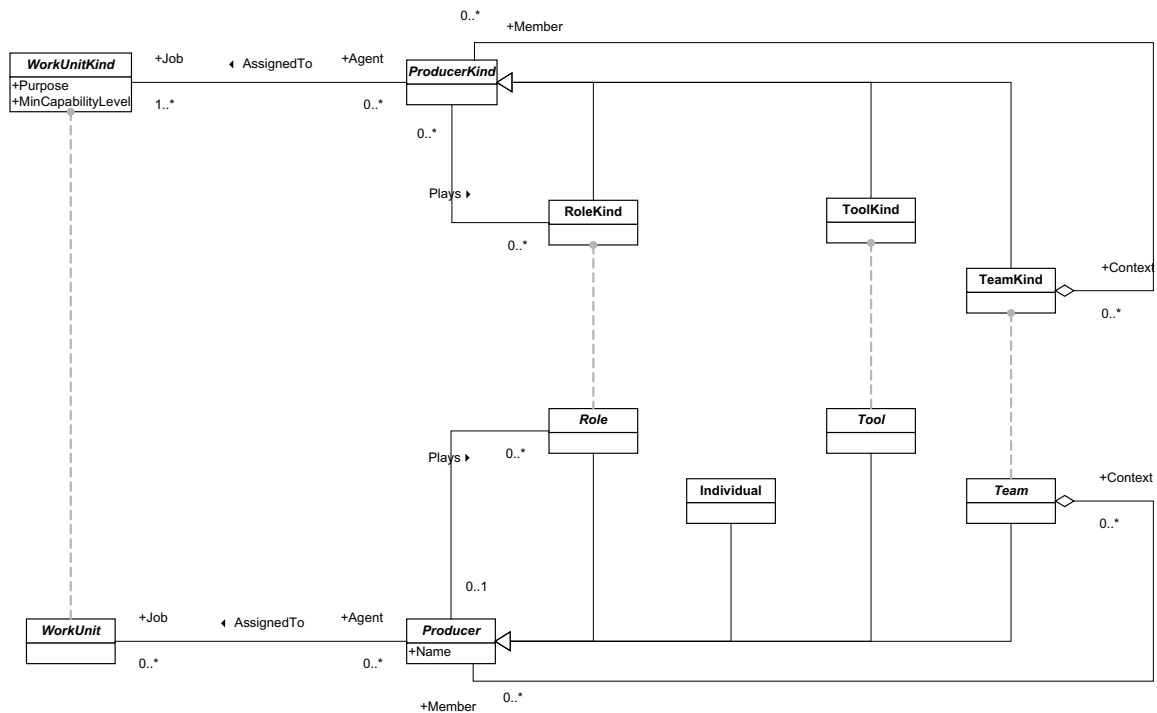


FIGURE 4 PRODUCERS

2.5 PRODUCT CLASSES

WorkProduct/*Kind has two subtypes: Model/*Kind (a mental representation of a fragment of the observed reality) and Document/*Kind (a durable depiction of a fragment of the observed reality). Although documents usually would depict models, they can also depict other entities of interest or even other documents. In fact, Document/*Kind has an association to WorkProduct/*Kind to represent this fact. Also, Document/*Kind has a recursive whole/part relationship with itself so a given work product or work product kind can be defined as a collection of other work products or work product kinds, respectively. See Figure 5 for a graphical depiction.

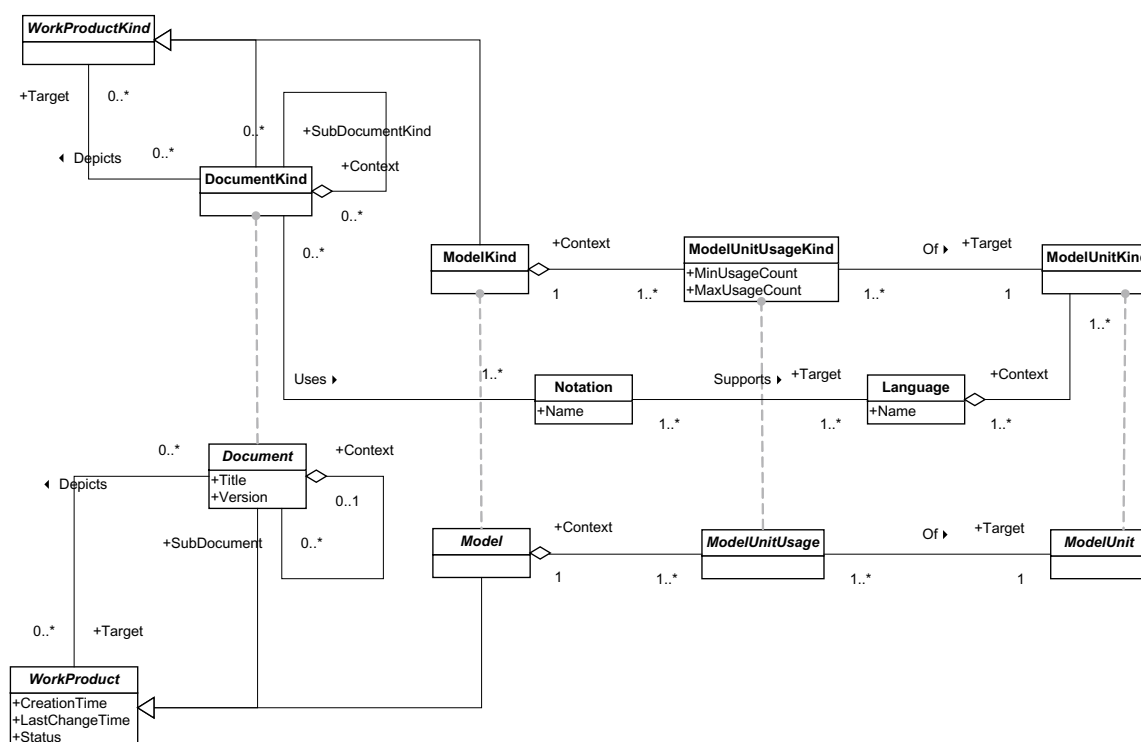


FIGURE 5 WORK PRODUCT AND MODELLING CLASSES

Model/*Kind, in turn, holds a whole/part relationship to ModelUnitUsage/*Kind, which in turn is associated to a specific ModelUnit/*Kind. This chain of relationships makes it possible to describe what model units are used in which models, and how they are employed. In addition, every ModelUnitKind is always defined in the context of at least one Language. Different languages sharing the same model unit kinds allow for a single and interconnected network of model units and models across a system rather than having different separate, isolated models. In turn, Language is associated with Notation to represent the fact that different notations support (or can depict) different languages. Finally, Notation is also related to DocumentKind to represent that each document kind makes use of at least one notation.

2.6 CONNECTION BETWEEN PROCESS AND PRODUCT

The interaction between the process and the product sides of the metamodel is achieved by the powertype pattern Action/*Kind. An Action/*Kind is always performed in the context of a given Task/*Kind (process side), and acts upon a given WorkProduct/*Kind (product side). The ActionKind.Type attribute takes values showing whether actions of a specific kind create, modify or only read the involved work product. See Figure 6 for a graphical depiction.

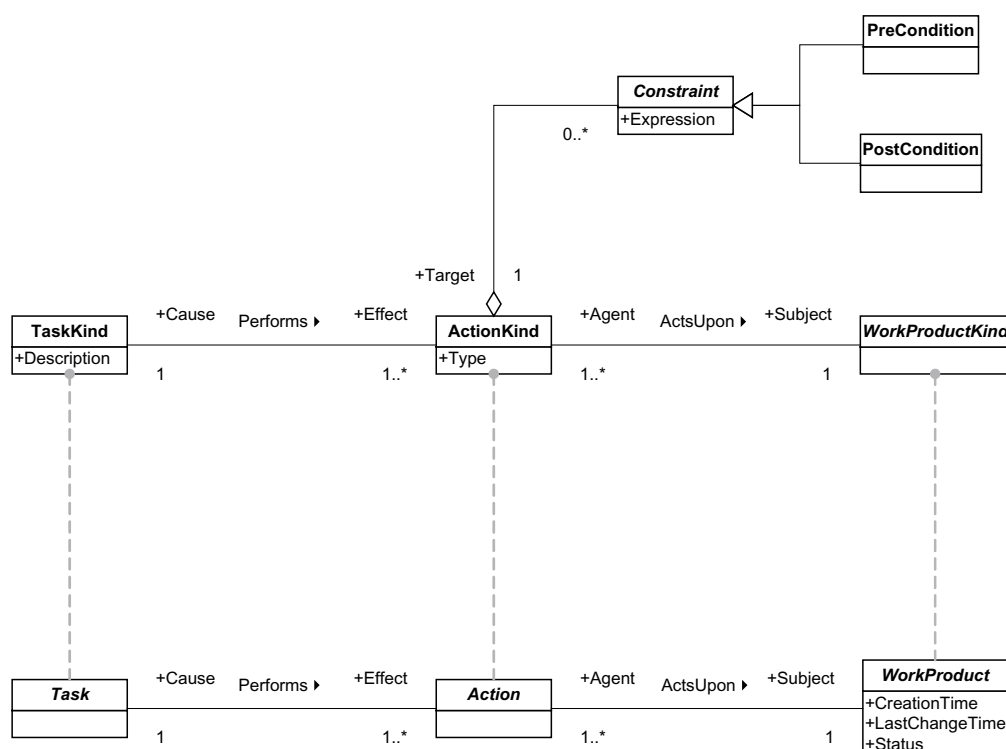


FIGURE 6 ACTIONS AND CONSTRAINTS

ActionKind is also related to Constraint, which is specialized into PreCondition (a condition that must hold in order for the associated action to proceed) and PostCondition (a condition that is guaranteed to hold after the associated action has been successfully performed).

2.7 SUPPORT CLASSES

In addition to the classes necessary to construct methodologies, some support classes exist for the convenience of methodologists using the SMSDM. See Figure 7 for a graphical depiction.

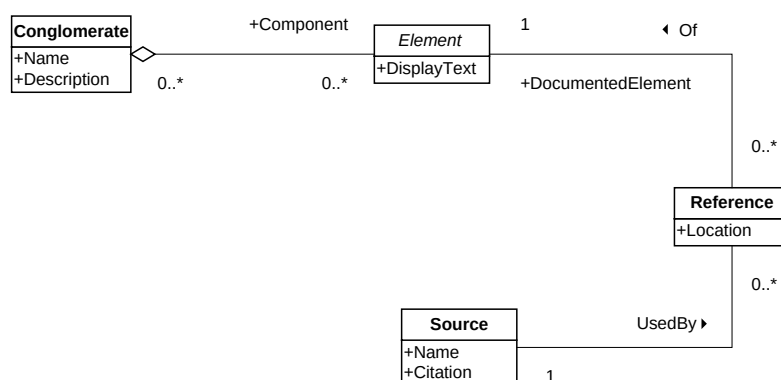


FIGURE 7 SUPPORT CLASSES

Some mechanism for specifying reusable methodology chunks is necessary. A Conglomerate class is defined to represent pre-defined collections of related elements (i.e. instances of the Element class) that can be reused in different methodological contexts.

Also, some means of managing references to bibliographic sources and best practices is needed. The Source class represents literature items or other sources of information and experience that a methodologist may want to use while constructing a methodology. The Reference class acts as a relationship between Source and Element so any number of linkages between elements and sources can be specified.

SECTION 3 ELEMENTS OF SMSDM

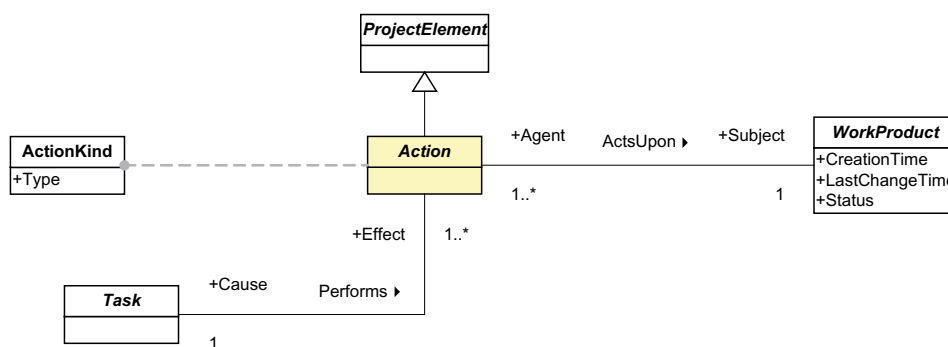
3.1 CLASSES IN THE SMSDM

In this section, the classes in the SMSDM are described in alphabetical order, and the attributes and relationships for each class are listed.

3.1.1 Action

An action is *a usage event performed by a task upon a work product*. Actions describe how specific tasks use specific work products, including the specification of the nature of such usage, i.e. creation, modification, etc.

This is an abstract subclass of Project Element.



The following applies:

(a) Attributes

This class has no attributes of its own.

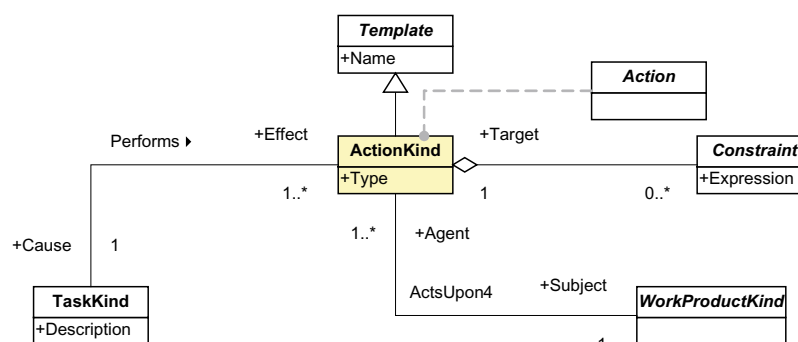
(b) Relationships

Name/Role	To class	Description
IsClassifiedBy	ActionKind	An action in the project layer is always of some action kind defined in the methodology layer.
ActsUpon	WorkProduct	An action always acts upon a particular work product.
Effect	Task	An action is always the effect of a particular task.

3.1.2 ActionKind

An action kind is *a specific kind of action, characterized by a given cause (a task kind), a given subject (a work product kind) and a particular type of usage*.

This is a subclass of Template.



The following applies:

(a) Attributes

Name	Type	Description
Type	ActionType	The nature of the usage that the associated task kind performs on the associated work product kind.

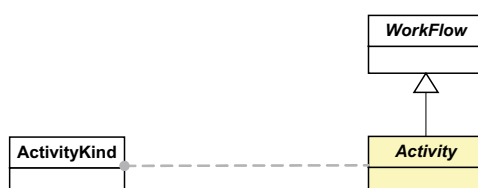
(b) Relationships

Name/Role	To class	Description
Classifies	Action	An action in the project layer is always of some action kind defined in the methodology layer.
ActsUpon	WorkProductKind	An action kind is always the agent that acts upon a particular work product kind.
Effect	TaskKind	An action kind is always the effect of a particular task kind.
Target	Constraint	An action kind may be constrained by some constraints.

3.1.3 Activity

An activity is *a work flow that represents a continuous responsibility*. Activities are generally used to represent infrastructural or supporting ongoing jobs.

This is an abstract subclass of Workflow.



The following applies:

(a) Attributes

This class has no attributes of its own.

(b) Relationships

Name/Role	To class	Description
IsClassifiedBy	ActivityKind	An activity in the project layer is always of some activity kind defined in the methodology layer.

3.1.4 ActivityKind

An activity kind is *a specific kind of activity, characterized by the area of expertise in which it occurs*.

This is a subclass of WorkflowKind.



The following applies:

(a) Attributes

This class has no attributes of its own.

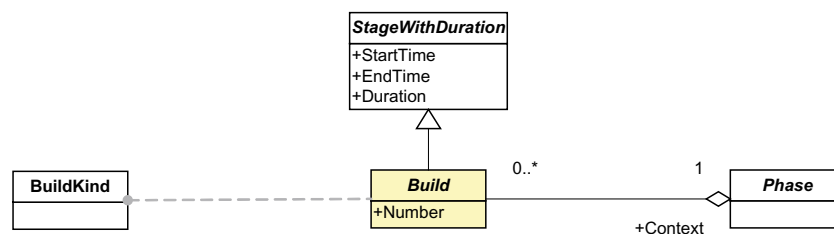
(b) Relationships

Name/Role	To class	Description
Classifies	Activity	An activity in the project layer is always of some activity kind defined in the methodology layer.

3.1.5 Build

A build is *a stage with duration for which the objective is the delivery of an incremented version of an already existing set of work products*. Builds are often used to implement incremental, iterative life cycles.

This is an abstract subclass of StageWithDuration.



The following applies:

(a) Attributes

Name	Type	Description
Number	String	The number of the build. Since builds are, by definition, incremental, some kind of numbering system is strongly recommended.

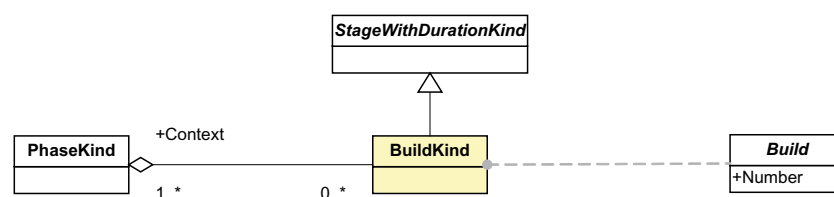
(b) Relationships

Name/Role	To class	Description
IsClassifiedBy	BuildKind	A build in the project layer is always of some build kind defined in the methodology layer.
n/a	Phase	A build always occurs in the context of a given phase.

3.1.6 BuildKind

A build kind is *a specific kind of build, characterized by the type of result that it aims to produce*.

This is a subclass of StageWithDurationKind.



The following applies:

(a) Attributes

This class has no attributes of its own.

(b) Relationships

Name/Role	To class	Description
Classifies	Build	A build in the project layer is always of some build kind defined in the methodology layer.
n/a	PhaseKind	A build kind is defined to occur in the context of some specific phase kinds.

3.1.7 Conglomerate

A conglomerate is *a collection of related elements that can be reused in different methodological contexts*. Conglomerates provide the basic reuse mechanism in SMSDM.



The following applies:

(a) Attributes

Name	Type	Description
Name	String	The name of the conglomerate.
Description	String	The description of the conglomerate, usually explaining the methodological contexts for which it has been designed.

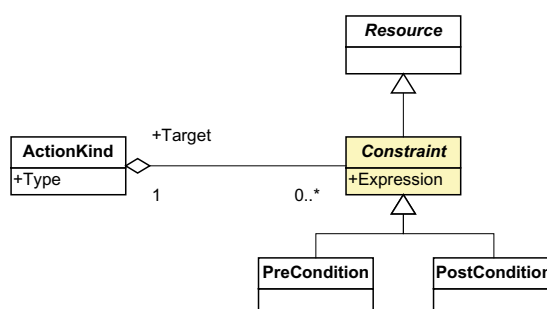
(b) Relationships

Name/Role	To class	Description
n/a	Element	A conglomerate is composed of a collection of elements.

3.1.8 Constraint

A constraint is *a condition that holds or must be held at certain point in time*. Constraints are often used to declaratively characterize the entry and exit conditions of actions.

This is an abstract subclass of Resource, specialized into PreCondition and PostCondition.



The following applies:

(a) Attributes

Name	Type	Description
Expression	String	The expression that must evaluate delete for the constraint to hold. Note that this is an abstract attribute and therefore different subclasses attach different specific semantics to it.

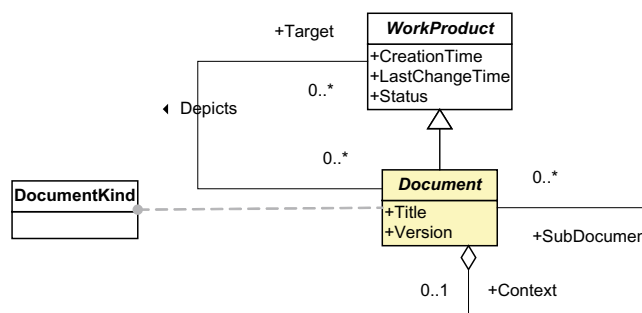
(b) Relationships

Name/Role	To class	Description
n/a	ActionKind	A constraint always characterizes a given action kind.

3.1.9 Document

A document is *a durable depiction of a fragment of the observed reality*. Documents often represent models, but they can also represent other subjects.

This is an abstract subclass of WorkProduct.



The following applies:

(a) Attributes

Name	Type	Description
Title	String	The title of the document.
Version	String	The version identifier of the document. Since documents are durable by definition, some version control is strongly recommended.

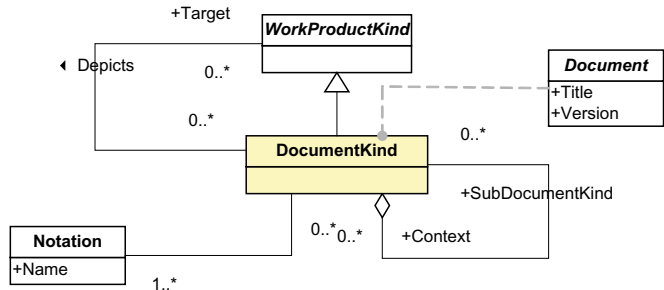
(b) Relationships

Name/Role	To class	Description
IsClassifiedBy	DocumentKind	A document in the project layer is always of some document kind defined in the methodology layer.
Context	Document	A document may be the context for sub-documents.
SubDocument	Document	A document may be a sub-document of a context document.
Depicts	WorkProduct	A document may depict a number of work products.

3.1.10 DocumentKind

A document kind is *a specific kind of document, characterized by its structure, type of content and purpose.*

This is a subclass of WorkProductKind.



The following applies:

(a) Attributes

This class has no attributes of its own.

(b) Relationships

Name/Role	To class	Description
Classifies	Document	A document in the project layer is always of some document kind defined in the methodology layer.
Context	DocumentKind	A document kind may be the context for sub-document kinds.
SubDocumentKind	DocumentKind	A document kind may be a sub-document kind of a number of context document kinds.
Depicts	WorkProductKind	A document kind may be defined to depict a number of work product kinds.
n/a	Notation	A document kind uses some given notations.

3.1.11 Element

An element is *a highly cohesive, loosely coupled entity of interest to the metamodel.* Since the SMSDM addresses both the methodology and the project layers (see Appendix A, Paragraph A2.3) any entity in either of these realms is subject to being modelled by the SMSDM and therefore to becoming represented by an element.

This is an abstract class, specialized into MethodologyElement and ProjectElement.

