Date: January, 2013

OBJECT MANAGEMENT GROUP

Case Management Model and Notation (CMMN)

FTF Beta 1

OMG Document Number: dtc/2013-01-01
Normative reference: http://www.omg.org/spec/CMMN/1.0
Machine readable file(s): http://www.omg.org/spec/CMMN/121101
Normative:
XMI:
http://www.omg.org/spec/CMMN/121101/CMMN10.xmi

XSD:
http://www.omg.org/spec/CMMN/121101/CMMN10.xsd
http://www.omg.org/spec/CMMN/121101/CMMN10CaseModel.xsd

This OMG document replaces the submission document (bmi/12-11-05, Alpha). It is an OMG Adopted Beta specification and is currently in the finalization phase. Comments on the content of this document are welcome, and should be directed to issues@omg.org by September 6, 2013.

You may view the pending issues for this specification from the OMG revision issues web page http://www.omg.org/issues/.

The FTF Recommendation and Report for this specification will be published on December 20, 2013. If you are reading this after that date, please download the available specification from the OMG Specifications web page http://www.omg.org/spec/.

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 109 Highland Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue (http://www.omg.org/report_issue.htm.)

Table of Contents

Table of Figures

Table of Tables

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at:
http://www.omg.org/spec

Specifications are organized by the following categories:

Business Modeling Specifications

Middleware Specifications

- **CORBA/IIOP**
- **Data Distribution Services**
- **Specialized CORBA**

IDL/Language Mapping Specifications

Modeling and Metadata Specifications

- **UML, MOF, CWM, XMI**
- **UML Profile**

Modernization Specifications

Platform Independent Model (PIM), Platform Specific Model (PSM), Interface Specifications

- **CORBAServices**
- **CORBAFacilities**

OMG Domain Specifications

CORBA Embedded Intelligence Specifications

CORBA Security Specifications

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
109 Highland Avenue
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 11 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier/Courier New - 11 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

NOTE: Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

1 Scope

1.1 Project Goals

The goals of this project are to:

1. Create a Case Management specification (referred to herein as CMMN 1.0).
2. Submit the specification as a response to the OMG RFP for Case Management Process Modeling (OMG document: bmi/2010-09-23)

1.2 In Scope

1. Define Case Management Model, notation, and operational semantics of the model.
2. Leverage a content management model based on properties, documents, folders, and relationships.
3. Leverage a standard expression and query language as the default CMMN expression and query language.
4. Specify how case events and constraints may be applied.
5. Specify interchange format for Case Management Model (XMI and XSD).

1.3 Out of Scope

1. Deployment mechanics
2. Run-time API's.
3. Operational simulation.
4. Definition of an [business] organizational model.
5. Definition of business goals.

2 Conformance

CM 1.0 shall conform to the relevant aspects of OMG BPMN 2.0 and the relevant aspects of OASIS CMIS 1.0.

3 References

3.1 Normative

RFC-2119

- Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, IETF RFC 2119, March 1997 http://www.ietf.org/rfc/rfc2119.txt

3.2 Non-Normative

Aalst, W.M.P. van der, Weske, M.: *Case Handling: a new paradigm for business process support*. Data & Knowledge Engineering. 53(2). 129-162, 2005

Davenport, Th. H. and Nohria, N., *Case Management and the Integration of Labor*, Sloan Management Review, 1994.

Davenport, Th. H, *Thinking for a Living: How to Get Better Performance and Results from Knowledge Workers*, HarvardBusinessSchool Press, 2005.

Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath III, F.T., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., and Vaculin, R., *Introducing the Guard-Stage-Milestone Approach for Specifying Business Entity Lifecycles*. Proceedings of the 7th International conference on Web Services and Formal Methods (WS-FM 2010). Bravetti, M., and Bultan, T. (eds.). Springer-Verlag, Berling, Heidelberg, 1-24. 2010.

Hull, R. et. al., *Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events*. Proceedings of the 5th ACM Intl. Conf. on Distributed Event-based Systems (DEBS), pages 51–62, New York, NY, USA, 2011.

Man, H. de, *Case Management: A Review of Modeling Approaches*, BPTrends, January 2009. [http://www.bptrends.com/publicationfiles/01-09-ART-%20Case%20Management-1-DeMan.%20doc--final.pdf]

4 Additional Information

4.1 Background

This specification defines a common meta-model and notation for modeling and graphically expressing a `Case`, as well as an interchange format for exchanging `Case` models among different tools. The specification is intended to capture the common elements that `Case` management products use, while also taking into account current research contributions on `Case` management. It is to `Case` management products what the OMG Business Process Model and Notation (BPMN) specification is to business process management products.

BPMN has been adopted as a business process modeling standard. It addresses capabilities incorporated in a number of other business process modeling languages, where processes are described as the predefined sequences of activities with decisions (gateways) to direct the sequence along alternative paths or for iterations. These models are effective for predefined, fully specified, repeatable business processes.

For some time, there has been discussion of the need to model activities that are not so predefined and repeatable, but instead depend on evolving circumstances and ad hoc decisions by knowledge workers regarding a particular situation, a *case* (see Davenport 1994 and 2005; and Van der Aalst 2005). Applications of `Case` management include licensing and permitting in government, application and claim processing in insurance, patient care and medical diagnosis in healthcare, mortgage processing in banking, problem resolution in call centers, sales and operations planning, invoice discrepancy handling, maintenance and repair of machines and equipment, and engineering of made-to-order products.

4.2 General concept

A `Case` is a proceeding that involves actions taken regarding a subject in a particular situation to achieve a desired outcome. Traditional examples come from the legal and medical worlds, where a legal `Case` involves the application of the law to a subject in a certain fact situation, and a medical `Case` involves the care of a patient in the context of a medical history and current medical problems. The subject of a `Case` may be a person, a legal action, a business transaction, or some other focal point around which actions are taken to achieve an objective. The situation commonly includes data that inform and drive the actions taken in a `Case`.

Any individual `Case` may be resolved in a completely ad-hoc manner, but as experience grows in resolving similar `Cases` over time, a set of common practices and responses can be defined for managing `Cases` in a more rigorous and repeatable manner. This becomes the practice of `Case` management, around which software products have emerged to assist `Case` Workers whose job is to process and resolve `Cases`.

Case management is often directed by a human—a Case manager or a team of Case workers—with minimal predefined encoding of the work to be performed. A Case may not have a single, designated Case manager, but may collaboratively engage different participants as required to make decisions or perform certain Tasks.

Planning at run-time is a fundamental characteristic of Case management. Case management requires modeling and notation which can express the essential flexibility that human Case workers, especially knowledge workers, require for run-time planning for the selection of Tasks for a Case, run-time ordering of the sequence in which the Tasks are executed, and ad-hoc collaboration with other knowledge workers on the Tasks (see De Man, January 2009).

Case management planning is typically concerned with determination of which Tasks are applicable, or which follow-up Tasks are required, given the state of the Case. Decisions may be triggered by events or new facts that continuously emerge during the course of the Case, such as the receipt of new documents, completion of certain Tasks, or achieving certain Milestones. Individual Tasks that are planned and executed in the context of the Case might be predefined procedural Processes in themselves, but the overall Case cannot be orchestrated by a predefined sequence of Tasks.

Representation of the circumstances and the decision factors in a Case model requires references to data about the subject of the Case. The collection of data about the Case is often described as a CaseFile. Documents and other unstructured or structured data about a Case are captured and referenced in the CaseFile for decision-making by Case workers.

Modeling of constraints and guidance on the actions to be taken in a Case requires the specification of rules that reference the data in the CaseFile. A Case model may specify constraints on run-time state transitions as well as constraints on actions, and recommendations for actions, that are dependent on the run-time state of the Case. Even though this specification is focused on modeling and notation, not run-time Case management per se, execution semantics is important for modeling of constraints and rules that depend on run-time state. To that end, execution semantics defined in this specification -- describing how EventListeners, Stages, Tasks, and Milestones affect each other and the state of the Case during the run-time management of a Case -- have been influenced by recent research into business artifacts and the guard-stage-milestone formalism (see Hull 2010).

Cases are directed not just by explicit knowledge about the particular Case and its context represented in the CaseFile, but also by explicit knowledge encoded as rules by business analysts, the tacit knowledge of human participants, and tacit knowledge from the organization or community in which participants are members.

A Case has two distinct phases, the design-time phase and the run-time phase. During the design-time phase, business analysts engage in modeling, which includes defining Tasks that are always part of pre-defined segments in the Case model, and “discretionary” Tasks that are available to the Case worker, to be applied in addition, to his/her discretion. In the run-time phase, Case workers execute the plan, particularly by performing Tasks as planned, while the plan may continuously evolve, due to the same or other Case workers being engaged in planning, i.e. adding discretionary Tasks to the plan of the Case instance in run-time. The following figure describes these concepts.

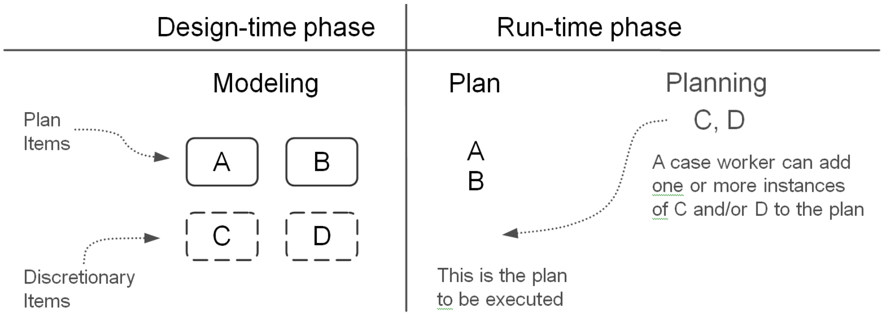

Design-time phase
Run-time phase
Modeling
Plan
Planning
Plan Items
A
B
C
D
Discretionary Items
A
B
C, D
A case worker can add one or more instances of C and/or D to the plan
This is the plan to be executed

Figure 1: Design-time phase modeling and run-time phase planning

4.3 Target users

Business analysts are the anticipated users of `Case` management tools for capturing and formalizing repeatable patterns of common `Tasks`, `EventListeners`, and `Milestones` into a `Case` model. A new `Case` model may be defined as entirely at the discretion of human participants initially, but it should be expected to evolve as repeatable patterns and best practices emerge. Patterns and outcomes from execution of the `Case` model can be incorporated iteratively by business analysts into the `Case` model, in the form of improved rules and more predictable patterns of `Tasks`, in order to make `Case` management more repeatable and improve outcomes over time.

4.4 Interoperability

In the context of `Case` management, this specification defines a meta-model (that is, a model for defining models), a notation for expressing `Case` models, and an XML Model for Interchange (XMI) and XML-Schema for exchanging `Case` models among different `Case` management vendors' environments and tools. The meta-model can be used by `Case` management definition tools to define functions and features that a business analyst could use to define a `Case` model for a particular type of `Case`, such as invoice discrepancy handling. The notation is intended for use by those tools to express the model graphically.

This specification enables portability of `Case` models, so that users can take a model defined in one vendor's environment and use it in another vendor's environment. The CMMN XMI and/or XML-Schema are intended for importing and exporting `Case` models among different `Case` management vendors' environments and tools.

A `Case` model is intended to be used by a run-time `Case` management product to guide and assist a knowledge worker in the handling of a particular instance of a `Case`, for example a particular invoice discrepancy. The meta-model and notation are used to express a case model in a common notation for a particular type of `Case`, and the resulting model can subsequently be instantiated for the handling of a particular instance of a `Case`.

4.5 Submitting and Supporting Organizations

The following companies are formal submitting members of OMG:

- BizAgi Limited
- Cordys Nederland BV
- International Business Machines Corporation
- Oracle Incorporated
- SAP AG
- Kofax plc

The following organizations have contributed to the development of this specification but are not formal submitters:

- Agile Enterprise Design, LLC
- Stiftelsen SINTEF
- TIBCO Software
- Trisotech

The following persons were members of the core teams that contributed to the content specification: Alan Babich, Henk de Man, Heidi Buelow, Bill Carpenter, Martin Chapman, Fred Cummins, Brian Elvesæter, Denis Gagne, Rick Hull, Dave Ings, Oliver Kieselbach, Matthias Kloppmann, Mike Marin, Greg Melahn, Paul O'Neill, Ralf Mueller, Ravi Rangaswamy, Jesus Sanchez, Arvind Srinivasan, Allen Takatsuka, Ivana Trickovic, Ganesh Vaideeswaran, Paul Vincent.

In addition, the following persons contributed valuable ideas and feedback that improved the content and the quality of this specification: Thomas Hildebrandt, Knut Hinkelmann, Jana Koehler, Matthias Kurz, Robert Lario, Paul Winsberg

4.6 IPR and Patents

The authors intend to contribute this work to OMG on a RF on RAND basis.

4.7 Guide to the Specification

This specification is organized into sections. Those sections that are normative are indicated as such.

5 Case Management Elements

5.1 Core Infrastructure

5.1.1 CMMNElement

`CMMNElement` is the abstract base class for all other classes in the `Case` metamodel.

Table 1: CMMNElement attributes

Attribute	Description
id : String	The ID of a `Case` metamodel object.
description : String	The description of a `Case` metamodel object

All reference associations between `CMMNElements` that are directly or indirectly contained in a `Case` MUST be resolvable within that `Case`, unless stated differently in the remainder of this specification.

5.1.2 Definitions

The `Definitions` class is the outermost containing object for all `CMMNElements`. It defines the scope of visibility and the namespace for all contained elements. The interchange of CMMN files will always be through one or more `Definitions`.

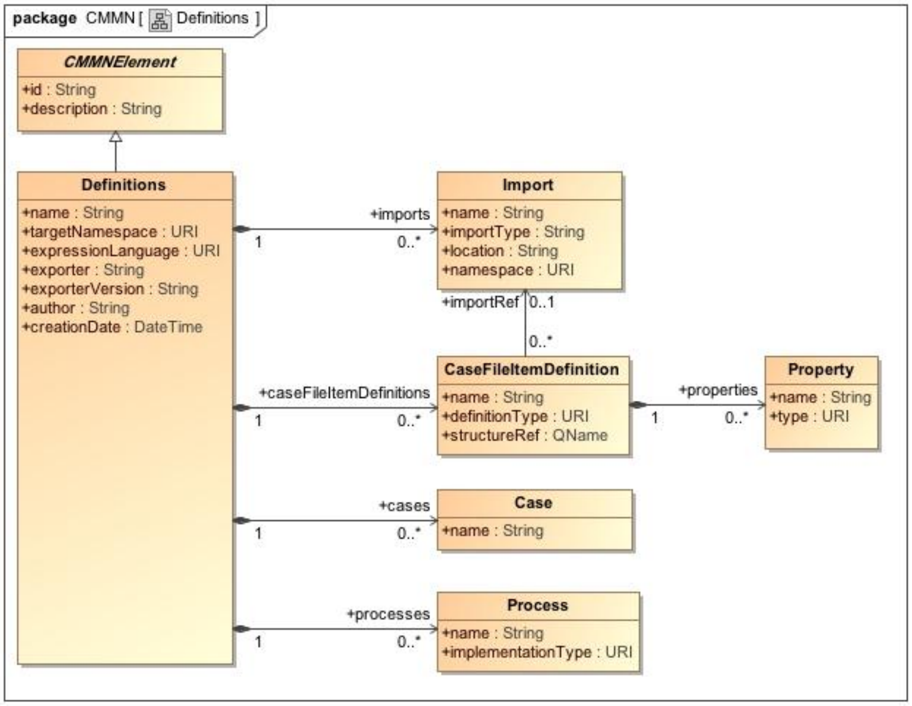


Figure 2: Definitions class diagram

defines the attributes of `Definitions`. It refers to concepts that are specified later on in the document, such as `Case` (5.2), `CaseFile` (5.3.1), `CaseFileItem` (5.3.2) and `Process` (5.4.8).

Table 2: Definitions attributes

Attribute	**Description**
name : String	The name of the Definitions object.
targetNamespace : URI	This attribute identifies the namespace associated with the Definitions objects and follows the convention established by XML Schema.
expressionLanguage : URI	The expression language used for this Definitions object. The default is "http://www.w3.org/1999/XPath". This value MAY be overridden on each individual Expression. The language MUST be specified in a URI format
exporter : String	This attribute identifies the tool that is exporting the CMMN model file.
exporterVersion : String	This attribute identifies the version of the tool that is exporting the CMMN model file
author : String	This attribute identifies the author of the CMMN model file
creationDate : DateTime	This attribute identifies the creation date of the CMMN model file
imports : Import[0..*]	This attribute is used to import externally defined elements and make them available for use by elements within this Definitions. A Definitions object that contains a Case MUST contain the Imports that are referenced by the CaseFileItemDefinitions of the CaseFileItems in the CaseFile of that Case.
caseFileItemDefinitions : CaseFileItemDefinition[0..*]	This attribute is used for the definition of CaseFileItem elements and makes those definitions available to use by elements within this Definitions. A Definitions object that contains a Case MUST contain the CaseFileItemDefinitions of the CaseFileItems in the CaseFile of that Case.
cases : Case[0..*]	This attribute is used to define Cases and make them available for use by elements within this Definitions.
processes: Process[0..*]	This attribute is used to define Processes and makes them available to use by elements within this Definitions. ProcessTasks of a Case

	MUST refer to Processes that are contained by the Definitions object that also contains the Case. ProcessTask and integration with Process is specified in 5.4.10.5.1.

5.1.3 Import

Type definitions that are externally defined can be imported into the CaseFile. This enables CaseFileItemDefinitions to refer to those externally defined types. The Import class has the following attributes:

Table 3: Import attributes

Attribute	Description
importType : String	The type of the import. For example, for XML-Schema, the import type is XSD.
location : String	The location URL of the import
namespace : URI	The namespace of the imported elements

For CaseFileItemDefinitions of definition type XSDElement, XSDComplexType, XSDSimpleType and XSDElement, the Import class SHOULD be used to import an XML Schema definition into the Case model. For other definition types, the use of Import is not further specified.

5.1.4 CaseFileItemDefinition

CaseFileItemDefinition elements specify the structure of a CaseFileItem. CaseFileItem is specified in 5.3.2.

Table 4: CaseFileItemDefinition attributes

Attribute	Description
definitionType : URI	The URI specifying the definition type of the CaseFileItem. specifies definition types.
structureRef : QName	A qualified name referring to the concrete structure of the definition entity. For XML-Schema typed case file definition elements, the structureRef refers to a XML complex type, element or simple type in a XML-Schema.
importRef : Import[0..1]	A (optional) reference to an Import. External structure definitions such as XML-Schema might be imported into the CaseFile and then referred from CaseFileItemDefinition
properties : Property[0..*]	Zero or more Property objects

The following definition types are specified for the CaseFileItemDefinition:

Table 5: DefinitionTypes and their URIs

Definition Type	URI

Folder in CMIS	http://www.omg.org/spec/CMMN/DefinitionType/CMISFolder
Document in CMIS	http://www.omg.org/spec/CMMN/DefinitionType/CMISDocument
Relationship in CMIS	http://www.omg.org/spec/CMMN/DefinitionType/CMISRelationship
XML-Schema Element	http://www.omg.org/spec/CMMN/DefinitionType/XSDElement
XML Schema Complex Type	http://www.omg.org/spec/CMMN/DefinitionType/XSDComplexType
XML Schema Simple Type	http://www.omg.org/spec/CMMN/DefinitionType/XSDSimpleType
Unknown	http://www.omg.org/spec/CMMN/DefinitionType/Unknown
Unspecified	http://www.omg.org/spec/CMMN/DefinitionType/Unspecified

5.1.4.1 Property

`Property` MAY complement `CaseFileItemDefinitions`. The following table gives an overview of the `Property` attributes:

Table 6: Property attributes

Attribute	Description
name : String	The name of the attribute
type : URI	The type of the attribute. The type MUST be a URI. specifies these types.

`Property types` are derived from the top-level built-in primitive types of XML Schema and include the following, see the description of the individual types in the XML Schema specification for an exact definition of the value space.

Table 7: Property Types and their URIs

Type	URI
string	http://www.omg.org/spec/CMMN/PropertyType/string
boolean	http://www.omg.org/spec/CMMN/PropertyType/boolean
integer	http://www.omg.org/spec/CMMN/PropertyType/integer
float	http://www.omg.org/spec/CMMN/PropertyType/float
double	http://www.omg.org/spec/CMMN/PropertyType/double
duration	http://www.omg.org/spec/CMMN/PropertyType/duration
dateTime	http://www.omg.org/spec/CMMN/PropertyType/dateTime
time	http://www.omg.org/spec/CMMN/PropertyType/time

date	http://www.omg.org/spec/CMMN/PropertyType/date
gYearMonth	http://www.omg.org/spec/CMMN/PropertyType/gYearMonth
gYear	http://www.omg.org/spec/CMMN/PropertyType/gYear
gMonthDay	http://www.omg.org/spec/CMMN/PropertyType/gMonthDay
gDay	http://www.omg.org/spec/CMMN/PropertyType/gDay
gMonth	http://www.omg.org/spec/CMMN/PropertyType/gMonth
hexBinary	http://www.omg.org/spec/CMMN/PropertyType/hexBinary
base64Binary	http://www.omg.org/spec/CMMN/PropertyType/base64Binary
anyURI	http://www.omg.org/spec/CMMN/PropertyType/anyURI
QName	http://www.omg.org/spec/CMMN/PropertyType/QName

5.2 Case Model Elements

`Case` is a top-level concept that combines all elements that constitute a `Case` model. The following diagram illustrates the metamodel of the `Case` and its associated classes.

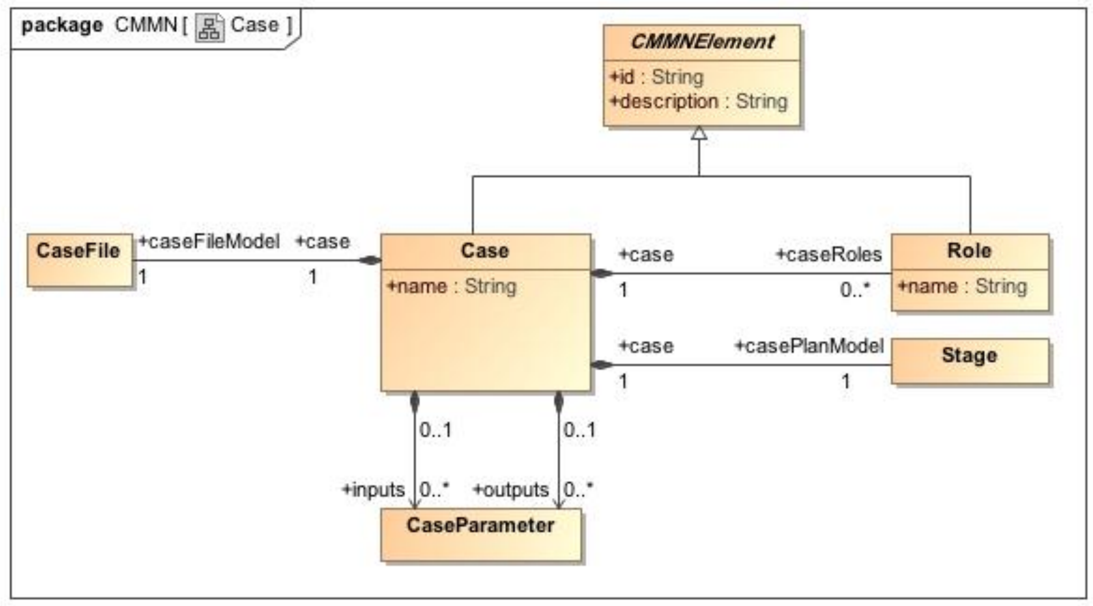


Figure 3: Case class diagram

A `Case` consists of a `caseFileModel`, a `casePlanModel` and a set of `caseRoles`. It also contains `inputs` and `outputs`, to enable interaction of the `Case` with its environment.

In this section we will regularly refer to aspects of CMMN execution semantics, in particular to elements of CMMN-defined lifecycles. Chapter 7 provides a complete specification of CMMN execution semantics and related lifecycles.

5.2.1 Case

The Case class inherits from the CMMNElement class and comprises of the following additional attributes:

Table 8: Case attributes

Attribute	Description
name : String	The name of the Case
caseRoles : Role[0..*]	This attribute lists the Role objects associated with the Case. These Roles are specific to the Case, and are not known outside the context of the Case.
caseFileModel : CaseFile[1]	One CaseFile object. Every Case MUST be associated with exactly one CaseFile. CaseFile is specified in 5.3.1.
casePlanModel : Stage[1]	The plan model of the Case. Every Case MUST be associated with exactly one plan model. It is defined by association to Stage. Stage is specified in 5.4.8. As it will appear in that section, Stage represents a recursive concept (Stages can be nested within other Stages), used as container of elements from which the plan of the case is constructed and can further evolve, and having a lifecycle that can be tracked in run-time. The "most outer" Stage is associated to the Case as its casePlanModel.
inputs : CaseParameter[0..*]	Input Parameters of the Case. A Case might have input Parameters so that it can be called from outside, e.g. by other Cases.CaseParameters are specified in 5.4.10.3.
outputs : CaseParameter[0..*]	Output Parameters of the Case. A Case might have output parameters so that it can return a result to e.g. a calling Case.

5.2.2 Role

CaseRoles authorize case workers or teams of case workers to perform HumanTasks (specified in 5.4.10.4), plan based on DiscretionaryItems (specified in 5.4.9.2), and raise user events (by triggering UserEventListeners, as specified in 5.4.2.2).

Example Roles of a case might be:

- Doctor. A doctor Role may contain one or more participants that are allowed to perform HumanTasks, trigger UserEventListeners, or do planning that requires doctor skills.
- Patient. A Case may provide an interface for patients to do planning that may correspond to scheduling appointments, complete HumanTasks that may correspond to providing information about their health, etc. In a typical application, a Case may limit the patient Role to contain a single participant.
- Nurse. A nurse Role may represent one or more participants with the skills of a nurse care provider

Assignment of Roles to participants, such as to individuals or teams, is not included in the scope of CMMN.

The Role class inherits from the CMMNElement class and comprises of the following additional attributes:

Table 9: Role attributes and model associations

Attribute	Description
name : String	The name of the Role
case : Case[1]	The Case that contains the caseRoles.

5.3 Information Model Elements

The information model of a Case comprises of classes for the management of the information (data) aspects of a Case. All information, or references to information, that is required as context for managing a Case, is defined by a CaseFile. The metamodel of CaseFile is represented in Figure 4.

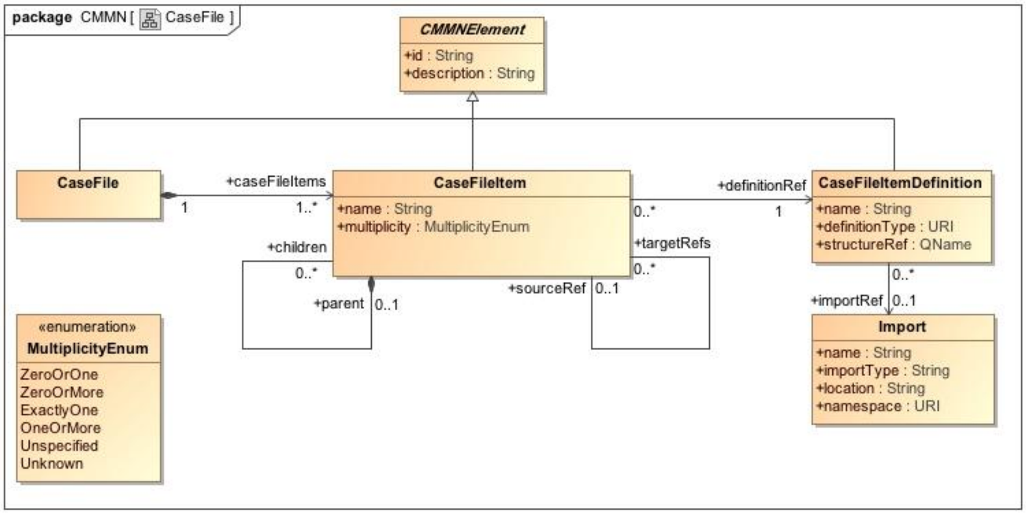


Figure 4: CaseFile class diagram

This model supports, amongst others, the information structure of the CMIS standard for content management systems, standards known from Service Oriented Architectures (SOA) like XML Schema and Object Oriented models based on UML.

5.3.1 CaseFile

Information in the CaseFile serves as context for raising events and evaluating Expressions as well as point of reference for CaseParameters, such as inputs and outputs of Tasks. CaseFile also serves as container for data that is accessible by other systems and people outside of the Case, through CaseParameters. CaseFile is meant as logical model. It does not imply any assumptions about physical storage of information.

Every Case is associated with exactly one CaseFile. The Case information is represented by the CaseFile. It contains CaseFileItems that can be any type of data structure. In particular containment hierarchies and other content objects can be represented. The Case File is represented in the metamodel by the class CaseFile, which has the following attributes:

Table 10: CaseFile attributes and model associations

Attribute	Description
caseFileItems : CaseFileItem[1..*]	This attribute lists the CaseFileItems of a CaseFile. A CaseFile MUST contain at least one CaseFileItem

5.3.2 CaseFileItem

A CaseFile consists of CaseFileItems. A CaseFileItem may represent a piece of information of any nature, ranging from unstructured to structured, and from simple to complex, which information can be defined based on any information modeling “language”. A CaseFileItem can be anything from a folder or document stored in CMIS, an entire folder hierarchy referring or containing other CaseFileItems or simply an XML document with a given structure. The structure, as well as the "language" (or format) to define the structure, is defined by the associated CaseFileItemDefinition (see 5.1.4). This may include the definition of properties ("metadata") of a CaseFileItem. If the internal content of the CaseFileItem is known, an XML Schema, describing the CaseFileItem, may be imported.

CaseFileItems can be used to represent containment structures organized into arbitrary hierarchies by using the parent/children containment association. For example, a folder hierarchy can be implemented by using a CaseFileItemDefinition.definitionType of CMISFolder, and using children and parent CaseFileItems as the folder structure. The resulting hierarchy can include metadata for each folder represented by the properties as defined by the associated CaseFileItemDefinition.

Case file items can be used to represent arbitrary content. For example, documents can be implemented by using CaseFileItemDefinition.definitionType of CMISDocument. There is no need to know the internals of those content objects, but if the internals of the object are known, the XML Schema can be defined by the Import class (see 5.1.3) of the CaseFileItemDefinition. The document or content object can include metadata as well, as represented by the properties as defined by the associated CaseFileItemDefinition.

The following attributes are defined for CaseFileItem:

Table 11: CaseFileItem attributes

Attribute	**Description**
name : String	The name of the CaseFileItem
multiplicity : MultiplicityEnum	The multiplicity of the CaseFileItem. The multiplicity specifies the number of potential instances of this CaseFileItem in the context of a particular Case instance. For example: An auto-damage claim might require "4" photographs of tire profiles. An antecedent investigation might involve "zero or more" police reports.
definitionRef : CaseFileItemDefinition[1]	A reference to the CaseFileItemDefinition. Every CaseFileItem MUST be associated to exactly one CaseFileItemDefinition.
children : CaseFileItem[0..*]	Zero or more children of the CaseFileItem. The children objects are contained by the CaseFileItem. A CaseFileItem is said to be "nested" in another CaseFileItem, when the CaseFileItem is a one the children of another CaseFileItem, either directly, or recursively through even other CaseFileItems. The set of children of a CaseFileItem MUST NOT include that CaseFileItem or any CaseFileItem in which that CaseFileItem is nested.
parent : CaseFileItem[0..1]	Zero or one parent of the CaseFileItem.

targetRefs : CaseFileItem[0..*]	Zero or more references to target `CaseFileItems`.
sourceRef : CaseFileItem[0..1]	Zero or one source `CaseFileItem`.

5.3.2.1 Versioning

This specification does not define versioning of `CaseFileItem` instances. It is recognized that any information element may have various versions, but a version control mechanism is outside the scope of this specification. It is also recognized that vendors may use version control mechanisms in their products, and such extensions may not be interchangeable. However, to guarantee basic interchangeability, when no extensions are used, it is assumed that whenever a case model, or expression, references an information element, that reference MUST refer to the latest, most current, version of that information element.

5.4 Plan Model Elements

This section specifies `casePlanModel` (see Figure 3). For a particular `Case` model, `casePlanModel` comprises both all elements that represent the initial plan of the case, and all elements that support the further evolution of the plan through run-time planning by case workers. As Figure 3 indicates, `casePlanModel` is defined by association to `Stage`. As it will appear in this section, `Stage` represents a recursive concept - `Stages` can be nested within other `Stages` - that serves as container of any element required to construct and further evolve `Case` plans. The “most outer” `Stage` is associated to the `Case` as its `casePlanModel`.

5.4.1 PlanItemDefinition

`PlanItemDefinition` elements define the building blocks from which `Case` (instance) plans are constructed. `PlanItemDefinition` is an abstract class that inherits from `CMMNElement`.

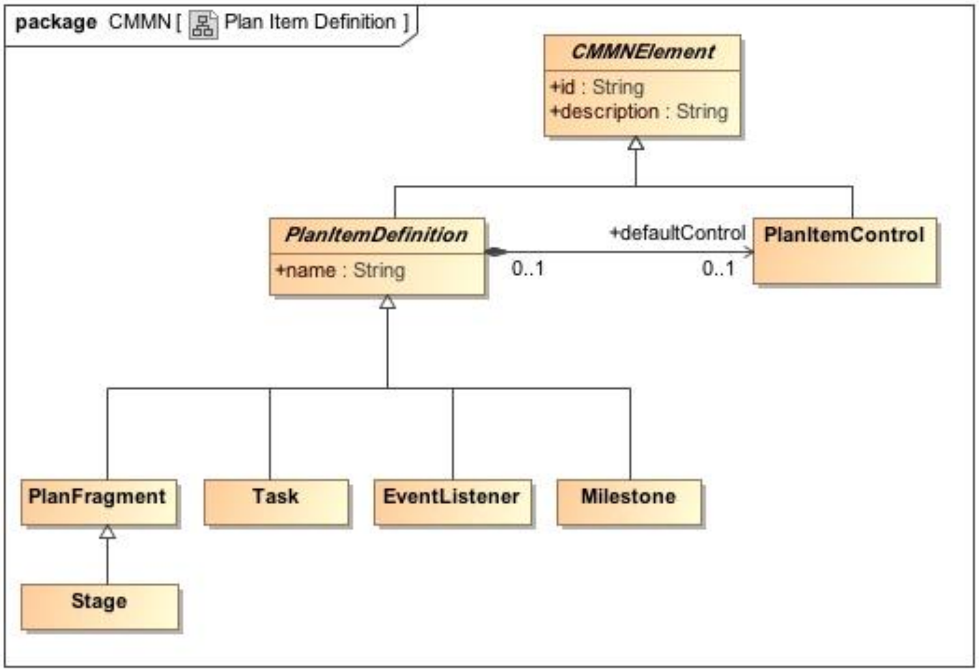


Figure 5: PlanItemDefinition class diagram

`PlanItemDefinition` is specialized into several concepts that are specified subsequent sections in this document: `EventListener`, `Milestone`, `PlanFragment` (and `Stage`) and `Task`.

The class PlanItemControl specifies defaults for aspects of control of PlanItemDefinitions, such as whether these instances have to be completed before the Case or a Stage of the Case, that contains the instances, can complete. PlanItemControl and these aspects will be specified in 5.4.11. As it will appear later, unlike Stages and the other sub-types of PlanItemDefinition, PlanFragments (that are not Stages) will not be instantiated in run-time.

PlanItemDefinition has the following attributes:

Table 12: PlanItemDefinition attributes

Attribute	Description
name : String	The name of the PlanItemDefinition
defaultControl : PlanItemControl[0..1]	Element that specifies the default for aspects of control of PlanItemDefinitions. DefaultControl MUST NOT be specified for the Stage that is referenced by the Case as its casePlanModel.

5.4.2 EventListener

In CMMN an event is something that “happens” during the course of a Case. Events may trigger, for example, the enabling, activation and termination of Stages and Tasks, or the achievement of Milestones. Any event has a cause. CMMN predefines many events, and their causes, such as:

- Anything that can happen to information in the CaseFile. This is defined by “standard events” that denote transitions in the CMMN-defined lifecycle of CaseFileItems.
- Anything that can happen to Stages, Tasks and Milestones. This is defined by “standard events” that denote transitions in the CMMN-defined lifecycle of these.

However, elapse of time cannot be captured via these “standard events”, and it will often lead to very indirect modeling, when any user event, such as the approval or rejection of something, has to be captured through impact on data in the CaseFile or through transitions in lifecyles of e.g. Tasks or Milestones.

For this reason, additional class is introduced, called EventListener, which is specialized into TimerEventListener and UserEventListener. EventListener has its own CMMN-predefined lifecycle, so that also any elapse of time as well as any user event, can still be captured as “standard events”, denoting transitions in the CMMN-defined lifecycle of EventListener.

EventListener inherits from PlanItemDefinition, so that instances of EventListeners can be elements of Case plans as well.

This will enable CMMN, to handle any event in a uniform way, namely as “standard events” that denote transitions in CMMN-defined lifecycles. These standard events are handled via Sentries. Sentries and these “standard events” are specified in section 5.4.6.

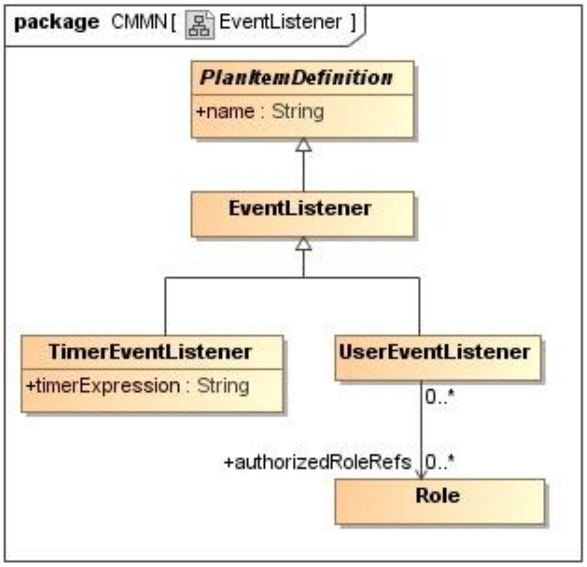


Figure 6: EventListener class diagram

5.4.2.1 TimerEventListener

A TimerEventListener is a PlanItemDefinition, which instances are used to catch predefined elapses of time. It inherits from EventListener. The following table lists the attributes of class TimerEventListener:

Table 13: TimerEventListener attributes

Attribute	Description
timerExpression : String	An expression string that is conforming to the ISO-8601 format for date and time, duration or interval representations.

5.4.2.2 UserEventListener

A UserEventListener is a PlanItemDefinition, which instances are used to catch events that are raised by a user, which events are used to influence the proceeding of the Case directly, instead of indirectly via impacting information in the CaseFile. A UserEventListener enables direct interaction of a user with the Case. It inherits from EventListener. The following table lists the attributes of class UserEventListener:

Table 14: UserEventListener attributes

Attribute	Description
authorizedRoleRefs : Role[0..*]	The Roles that are authorized to raise the user event

5.4.3 Milestone

A Milestone is a PlanItemDefinition that represents an achievable target, defined to enable evaluation of progress of the Case. No work is directly associated with a Milestone, but completion of set of tasks or the availability of key deliverables (information in the CaseFile) typically leads to achieving a Milestone.

5.4.4 PlanFragment

A PlanFragment is a set of PlanItems (see 5.4.5), possibly dependent on each other, and that often occur in Case plans in combination, representing a pattern.

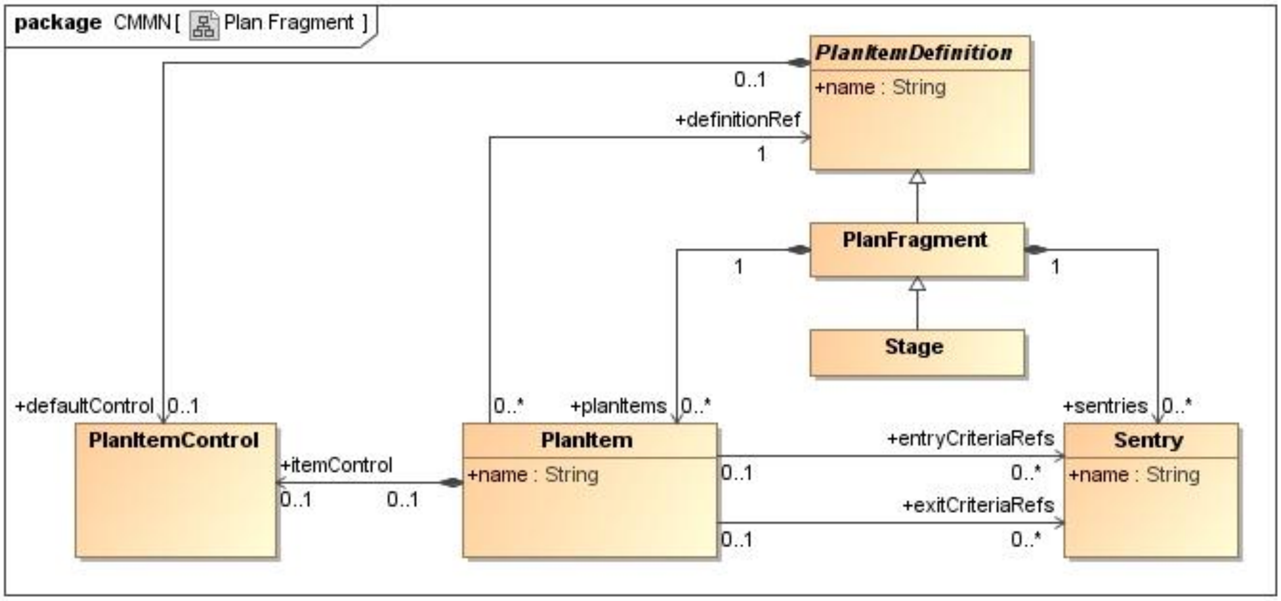


Figure 7: PlanFragment class diagram

Dependencies between PlanItems, in PlanFragments, are defined as Sentries (see 5.4.6). A PlanFragment is a container of PlanItems and the Sentries that define the criteria according to which the PlanItems are enabled (or entered) and terminated (or exited).

Simple examples of PlanFragments are:

- A combination of two Tasks, whereby, the completion of one Task satisfies the Sentry that enables the start of the other.
- A combination of an EventListener and a Task, whereby the occurrence of the event satisfies the Sentry that enables the start of the Task.

PlanFragments can represent PlanItem-and-Sentry patterns of any complexity. Simple PlanFragments may not contain Sentries. PlanFragment inherits from PlanItemDefinition, because the combination of PlanItems (and Sentries) that it contains can be added to the plan of a Case (instance) as a unit. Unlike other PlanItemDefinitions, a PlanFragment (that is not a Stage) does not have a representation in run-time, i.e. there is no notion of lifecycle tracking of a PlanFragment (not being a Stage) in the context of a Case instance. Just the PlanItems that are contained in it are instantiated and have their lifecyles that are tracked.

In order to plan a combination of PlanItems that is tracked, in the plan of a Case instance, as combination, a specialization of PlanFragment should be used, called a Stage. Stage is specified in 5.4.8. Stages have lifecycles, PlanFragments (not being Stages) don't.

The class PlanFragment has the following attributes:

Table 15: PlanFragment attributes and model associations

Attribute	Description
planItems : PlanItem[0..*]	The PlanItems that are contained by the PlanFragment.
sentries : Sentry[0..*]	The Sentrys contained by the PlanFragment.

5.4.5 PlanItem

A PlanItem object is a use of a PlanItemDefinition element in a PlanFragment (or Stage) .

As soon as experience is gained in applying a Case model, best practices might evolve, e.g. recognizing the usefulness, or even necessity, of applying re-usable combinations of PlanItemDefinitions. The same PlanItemDefinition might be (re-)used multiple times as part of different combinations, i.e. as part of different PlanFragments (or Stages). Hence, a PlanItemDefinition, e.g. a Task or EventListener, is defined once, and can be (re-) used in multiple PlanFragments (and Stages).

This required a separate class, PlanItem, that refers to PlanItemDefinition. Multiple PlanItems might refer to the same PlanItemDefinition. A PlanItemDefinition is (re-)used in multiple PlanFragments (or Stages) when these PlanFragments (or Stages) contain PlanItems that refer to or ("use") that same PlanItemDefinition.

Table 16: PlanItem attributes

Attribute	Description
name : String	The name of the PlanItem object. This attribute supersedes the attribute of the corresponding PlanItemDefinition element.
itemControl : PlanItemControl[0..1]	The PlanItemControl controls aspects of the behavior of instances of the PlanItem object. If a PlanItemControl object is specified for a PlanItem, then it MUST overwrite the PlanItemControl object of the associated PlanItemDefinition element. Otherwise, the behavior of the PlanItem object is specified by the PlanItemControl object of its associated PlanItemDefinition. PlanItemControl is specified in 5.4.11.
definitionRef : PlanItemDefinition[1]	Reference to the corresponding PlanItemDefinition object. For every PlanItem object, there MUST be exactly one PlanItemDefinition object. DefinitionRef MUST NOT represent the Stage that is the casePlanModel of the Case. DefinitionRef MUST NOT represent a PlanFragment that is not a Stage. This implies that a PlanFragment, not being a Stage, cannot be

	used as PlanItem inside a PlanFragment or Stage. As PlanItems may refer to a PlanItemDefinition that is a Stage, Stages can be nested. A Stage is said to be "nested" in another Stage, when the Stage is the PlanItemDefinition of a PlanItem that is contained in that other Stage, either directly, or recursively through even other Stages. DefinitionRef of a PlanItem that is contained by a Stage MUST NOT be that Stage or any Stage in which that Stage is nested.
entryCriteriaRefs : Sentry[0..*]	Reference to zero or more Sentries that represent the PlanItem's entry criteria. EntryCriteriaRefs of a PlanItem MUST refer to Sentries that are contained by the Stage or PlanFragment that contains that PlanItem. A PlanItem that is defined by an EventListener MUST NOT have entryCriteriaRefs.
exitCriteriaRefs : Sentry[0..*]	Reference to zero or more Sentries that represent the PlanItem's exit criteria. ExitCriteriaRefs of a PlanItem MUST refer to Sentries that are contained by the Stage or PlanFragment that contains that PlanItem. A PlanItem that is defined by an EventListener or Milestone MUST NOT have exitCriteriaRefs. A PlanItem that is defined by a Task that is non-blocking (isBlocking set to "false") MUST NOT have exitCreteriaRefs .

5.4.6 Sentry

A Sentry "watches out" for important situations to occur (or "events"), which influence the further proceedings of a Case (and hence their name).

A Sentry is a combination of an "event and/or condition". When the event is received, a condition might be applied to evaluate whether the event has effect or not. Sentries may take the following form:

1. An event part and a condition part in the form
   on <event> if <condition>
   or
2. An event part in the form
   on <event>
   or
3. Just a condition part in the form
   if <condition>

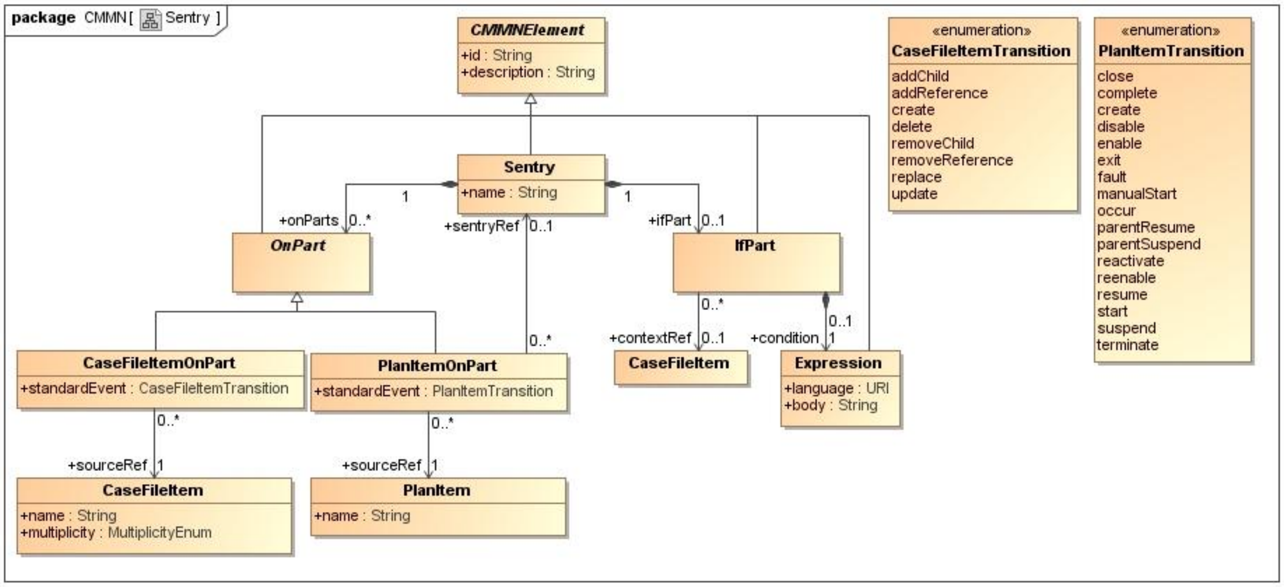


Figure 8: Sentry class diagram

As discussed in section 5.4.2, CMMN defines of a set of "standard events", based on transitions in CMMN-defined lifecycles, that is capable of capturing any event that is relevant in the context of a `Case`. This includes timer events, case information events and user events.

A `Sentry` may consist of two parts:

- One or more `OnParts`. An `OnPart` specifies the event that serves as trigger. When the event is catched, the `OnPart` is said to "occur".
- Zero or one `IfPart`. The `IfPart` specifies a condition, as `Expression` that evaluates over the `CaseFile`. If all `OnParts` of a `Sentry` have occurred, and its `IfPart` (if existent) evaluates to "true", the `Sentry` is said to be "satisfied".

A `Sentry` that is satisfied actually triggers the `PlanItem` that refers to it (see Figure 7):

- When the `Sentry` is referenced by one of the `PlanItem`'s e`ntryCriteriaRefs`, the `PlanItem` (its instance) will transit, based on the entry criteria-related transition in its lifecycle: a `Task` or `Stage` will be enabled, and a `Milestone` will be achieved.
- When the `Sentry` is referenced by one of the `PlanItem`'s `exitCriteriaRefs`, the `PlanItem` will transit, based on the exit criteria-related transition in its lifecycle: a `Task` or `Stage` will be terminated (exited).

Chapter 7 will analyze the relationship between `Sentries` and lifecycles in detail.

`Sentry` inherits from `CMMNElement` and has the following attributes:

Table 17: Sentry attributes

Attribute	Description
name : String	The name of the `Sentry`.
onParts : OnPart[0..*]	Defines the `OnParts` of the `Sentry`.

ifPart : IfPart[0..1]	Defines the IfPart of the Sentry.

A Sentry MUST have an IfPart or at least one OnPart.

5.4.6.1 OnPart

The Sentry OnPart addresses the "event" aspect of a Sentry. The class OnPart is an abstract class that inherits from CMMNElement. It has two sub-classes: CaseFileItemOnPart and PlanItemOnPart.

5.4.6.2 CaseFileItemOnPart

The class CaseFileItemOnPart inherits from OnPart and has the following attributes:

Table 18: CaseFileItemOnPart attributes

Attribute	Description
standardEvent : CaseFileItemTransition	Reference to a state transition in the CaseFileItem lifecyle (see 7.2). The enumeration CaseFileItemTransition is specified in 5.4.6.2.1.
sourceRef : CaseFileItem[1]	Reference to a CaseFileItem. If the associated CaseFileItem is undergoing the state transition as specified by attribute standardEvent, the OnPart MUST occur (in run-time).

5.4.6.2.1 CaseFileItemTransition

CaseFileItemTransition is an enumeration that specifies transitions in the CMMN-defined lifecycle of CaseFileItems (see 7.2). Its values are:

Table 19: CaseFileItemTransition enumeration

CaseFileItem Lifecycle State transition	Description
addChild	A new child CaseFileItem has been added to an existing CaseFileItem. The lifecycle state remains Available.
addReference	A new reference to a CaseFileItem has been added to a CaseFileItem. The lifecycle state remains Available.
create	A CaseFileItem transitions from the initial state to Available.
delete	A CaseFileItem transitions from Available to Discarded
removeChild	A child CaseFileItem has been removed from a CaseFileItem. The lifecycle state remains Available.
removeReference	A reference to a CaseFileItem has been removed from a CaseFileItem. The lifecycle state remains Available.

replace	The content of a CaseFileItem has been replaced. The lifecycle state remains Available.
update	The CaseFileItem has been updated. The lifecycle state remains Available.

5.4.6.3 PlanItemOnPart

The class PlanItemOnPart inherits from OnPart and has the following attributes:

Table 20: PlanItemOnPart attributes

Attribute	Description
standardEvent : PlanItemTransition	Reference to a state transition in the lifecycle of a Stage, Task, EventListener or Milestone (see 7.3). The enumeration PlanItemTransition is specified in 5.4.6.3.1. If definitionRef of the PlanItem, that is referenced by the OnPart as sourceRef, represents a Stage or Task, the value of standardEvent of the OnPart MUST denote a transition of the CMMN-defined lifecycle of Stage / Task (see 7.3.2). If definitionRef of the PlanItem, that is referenced by the OnPart as sourceRef, represents an EventListener or Milestone, the value of standardEvent of the OnPart MUST denote a transition of the CMMN-defined lifecycle of EventListener / Milestone (see 7.3.3).
sourceRef : PlanItem[0..1]	Reference to a PlanItem. If the associated PlanItem is undergoing a state transition as specified by attribute standardEvent the OnPart MUST occur (in run-time). SourceRef represents a PlanItem that MUST be contained by the same PlanFragment (or Stage) that also contains the Sentry that contains the PlanItemOnPart.
sentryRef: Sentry [0..1]	A reference to a Sentry. It enforces that the PlanItemOnPart of the Sentry occurs when the PlanItem that is referenced by sourceRef transits by the exit transition in its lifecycle, due to the Sentry that is referenced by sentryRef being satisfied. An example is provided and explained in section 6.10.1, in relation to Figure 42. SentryRef, if specified, MUST refer to a Sentry that is referenced by an exitCriteriaRef of the PlanItem that is referred to as the sourceRef of the PlanItemOnPart. When sentryRef is specified, standardEvent MUST have value “exit”.

5.4.6.3.1 PlanItemTransition

PlanItemTransition is an enumeration that specifies transitions in the CMMN-defined lifecycles of Stages, Tasks, EventListeners and Milestones (see 7.3). Its values are:

Table 21: PlanItemTransition enumeration

PlanItem Lifecycle State transition	**Description**
close	The casePlanModel transitions from Completed, Terminated, Failed or Suspended to Closed
complete	The casePlanModel, Stage or Task transitions from Active to Completed.
create	• The casePlanModel transitions from the initial state to Active • The PlanItem transitions from the initial state to Available
disable	The Stage or Task transitions from Enabled to Disabled
enable	The Stage or Task transitions from Available to Enabled
exit	The Stage or Task transitions from Available, Enabled, Disabled, Active, Failed or Suspended to Terminated.
fault	The Stage or Task transitions from Active to Failed
manualStart	The Stage or Task transitions from Enabled to Active.
occur	The EventListener or Milestone transitions from Available to Completed.
parentResume	The Stage or Task transitions from Suspended to Available, Enabled, Disabled or Active depending on its state before it was suspended
parentSuspend	The Stage or Task transitions from Available, Enabled, Disabled or Active to Suspended
reactivate	• The casePlanModel transitions from Completed, Terminated, Failed or Suspended to Active • The PlanItem transitions from Failed to Active
reenable	The Stage or Task transitions from Disabled to Enabled
resume	• The Task or Stage transitions from Suspended to Active. • The EventListener or Milestone transitions from Suspended to Available.
start	The Stage or Task transitions from Available to Active
suspend	• The casePlanModel, Stage or Task transitions from Active to Suspended. • The EventListener or Milestone transitions from Available to Suspended.
terminate	• The casePlanModel, Stage or Task transitions from Active to Terminated

	• The EventListener or Milestone transitions from Available to Terminated.

5.4.6.4 IfPart

The IfPart of a Sentry is used to specify an (optional) condition.

The class IfPart inherits from CMMNElement, and has the following attributes:

Table 22: IfPart attributes

Attribute	Description
contextRef : CaseFileItem[0..1]	The context of the IfPart. The caseFileItem that serves as starting point for evaluation of the Expression that is specified by the condition of the IfPart. If not specified, evaluation starts at the CaseFile object that is referenced by the Case as its caseFileModel.
condition : Expression[1]	A condition that is defined as Expression. The Expression MUST evaluate to boolean. Expressions are specified in 5.4.7.

5.4.7 Expressions

Expressions specify String objects that are evaluated over information in the CaseFile. Expressions do also specify the language in which the String objects MUST be specified.

Expression inherits from CMMNElement, and has the following attributes:

Table 23: Expression attributes

Attribute	Description
language : URI	The language in which the Expression body is specified. The language attribute is optional. The default value of the language attribute is defined by the value of expressionLanguage of the Definitions object. If a value is specified for the language attribute of an Expression, it overwrites the default for that Expression.
body : String	The actual expression. It MUST be valid according to the specified language.

5.4.8 Stage

A Stage inherits from PlanFragment. As PlanFragment it is a PlanItemDefinition as well, and serves as building block for Case (instance) plans therefore.

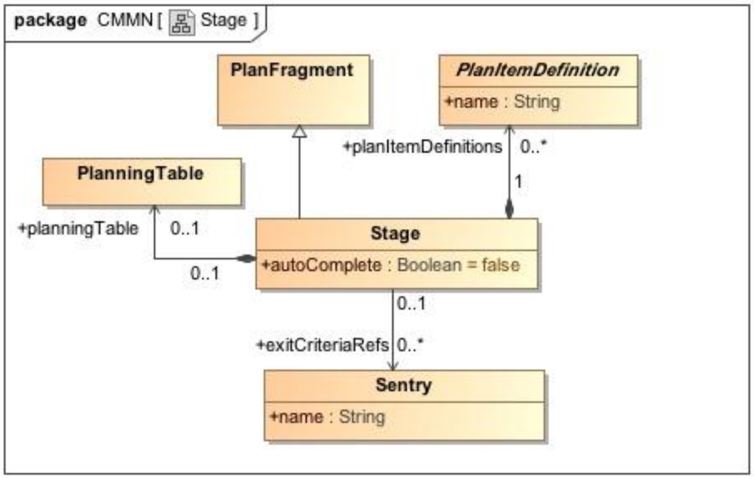


Figure 9: Stage class diagram

As a `PlanFragment`, a `Stage` can contain `PlanItems` and `Sentries`.

Unlike `PlanFragments` (that are not `Stages`), `Stages` do have run-time representations in a `Case` (instance) plan. Instances of `Stages` are tracked through the CMMN-defined `Stage` lifecycle (see 7.3.2). `Stages` maybe considered "episodes" of a `Case`, though `Case` models allow for defining `Stages` that can be planned in parallel also.

The following is supported for a `Stage`, which is not supported for a `PlanFragment` (that is not a `Stage`):

- A `Stage` can be used as `PlanItem` inside `PlanFragments` or other `Stages`.
- A `Stage` (instance) can serve as context for planning, i.e. a `Stage` can have a `PlanningTable`, to support users in planning additional ("discretionary") items into instances of the `Stage` in run-time. `PlanningTables` and `DiscretionaryItems` are specified in 5.4.9
- The `Case` refers to a `Stage` as its `casePlanModel`. This defines the "most outer" `Stage` of the `Case`.
    - This "most outer" `Stage` also contains the `PlanItemDefinitions` that are used in the `Case`.
    - This "most outer" `Stage` of the `Case` may also contain `Sentries` that serve as exit criteria for that `Stage`, and hence for the `Case`.

The class `Stage` has the following attributes:

Table 24: Stage attributes

Attribute	**Description**
planItemDefinitions : PlanItemDefinition[0..*]	This attribute lists the `PlanItemDefinition` objects available in the `Stage`, and its nested `Stages`. `PlanItemDefinitions` MUST NOT be contained by any other `Stage` than the `casePlanningModel` of the `Case`.
autoComplete : Boolean = false	This attribute controls completion of the `Stage`. If "false", a `Stage` requires a user to manually complete it, which is often appropriate for `Stages` that contain "discretionary"

	items (see 5.4.9.2) and/or non-required Tasks or Stages (see 5.4.11.2). Stage completion logic is specified in detail in 7.5.1.
planningTable : PlanningTable[0..1]	Defines the (optional) PlanningTable of the Stage. PlanningTable is specified in 5.4.9.
exitCriteriaRefs : Sentry[0..*]	Reference to zero or more Sentries that serve as the exit criteria for the Stage. ExitCriteriaRefs of a Stage MUST refer to Sentries that are contained by that Stage. Only the Stage that is referenced by the Case as its casePlanningModel can have exitCriteriaRefs. Note that it is only useful for that Stage to directly have exitCriteriaRefs, as it cannot be further nested in other Stages (other Stages can contain both PlanItems that represent Stages and the Sentries that impose entry and/or exit criteria on them).

5.4.9 PlanningTable

Planning is a run-time effort. A PlanningTable defines the scope of planning, in terms of identifying a sub-set of PlanItemDefinitions that can be considered for planning in a certain context. The context for planning might be:

- A Stage. When a Stage has a PlanningTable, that PlanningTable can be used, for an instance of that Stage, to plan instances of Tasks and Stages into that Stage instance.
- A HumanTask. When a HumanTask has a PlanningTable (see 5.4.10.4), that PlanningTable can be used, for an instance of that HumanTask, to plan instances of Tasks and Stages into the instance of the Stage that contains that instance of the HumanTask.

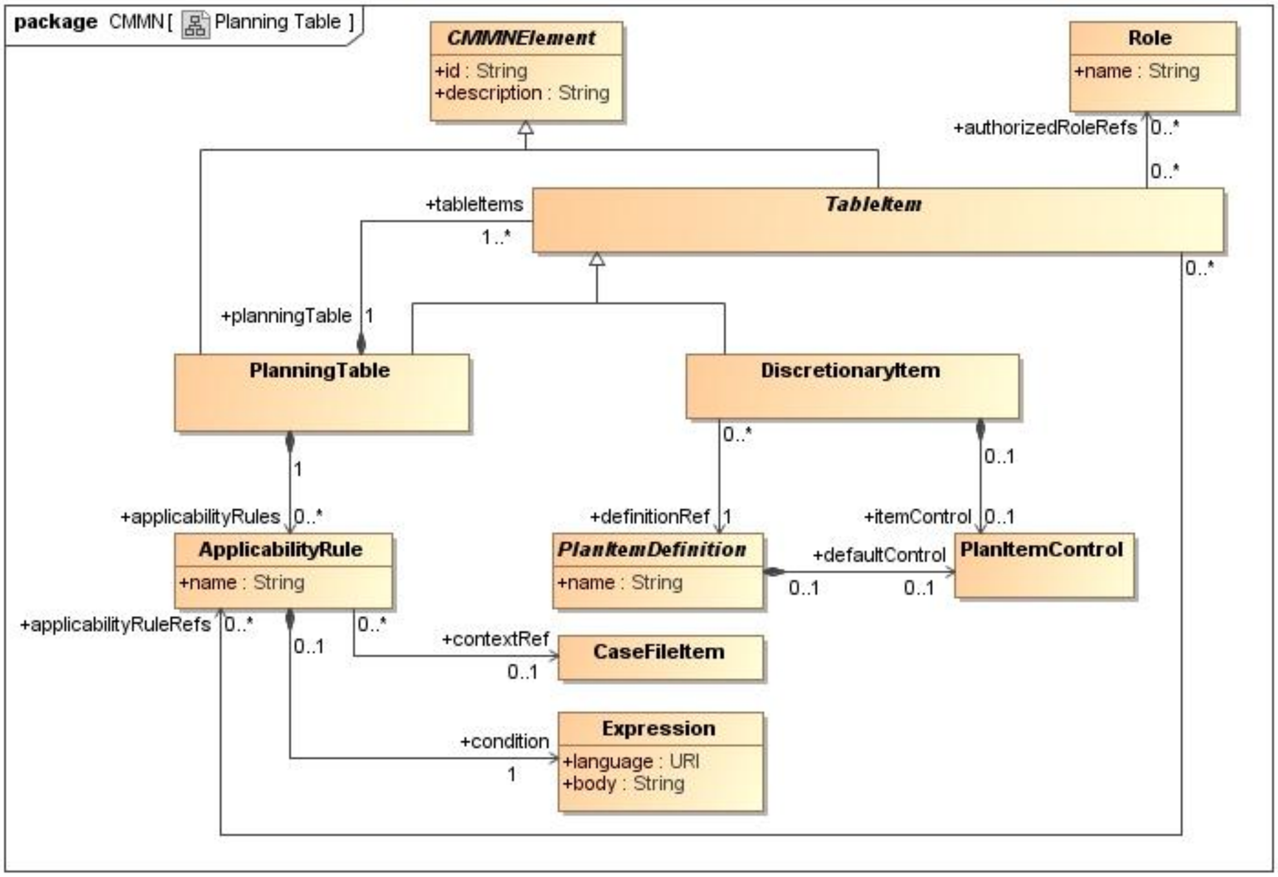


Figure 10: PlanningTable class diagram

Instances of Tasks and Stages that are defined by the same PlanItemDefinition might be planned based on possibly multiple PlanningTables. This required a separate class, DiscretionaryItem (see 5.4.9.2), that refers to PlanItemDefinition. Multiple DiscretionaryItems might refer to the same PlanItemDefinition. A PlanItemDefinition is (re-)used in multiple PlanningTables when these PlanningTables contain DiscretionaryItems that refer to or ("use") that same PlanItemDefinition.

For convenience, a DiscretionaryItem that refers to a PlanItemDefinition that is a Task, might be called a "discretionary Task". Similarly we can consider "discretionary PlanFragments" and "discretionary Stages". Note that PlanFragments that are no Stages can only be "discretionary", as PlanItems cannot refer to them (see 5.4.5). Note again that, when a PlanFragment (that is not a Stage) is used for planning, just the PlanItems that are contained in it are instantiated and have their lifecyles that are tracked. The PlanFragment (that is not a Stage) is not instantiated itself.

For convenience during run-time planning, in situations where a PlanningTable would contain potentially many DiscretionaryItems, it is possible to define a PlanningTable recursively: a PlanningTable containing other PlanningTables.

Users (Case workers) are said to "plan" (in run-time), when they select DiscretionaryItems from a PlanningTable, and move instances of their associated PlanItemDefinitions into the plan of the Case (instance).

It is possible to authorize Roles for planning of certain DiscretionaryItems and sub-PlanningTables. It is also possible to make DiscretionaryItems (and sub-PlanningTables) dynamically applicable for planning, based on conditions that evaluate over the CaseFile. Both Role authorizations and ApplicabilityRules (see 5.4.9.3) can dynamically control what DiscretionaryItems, possibly organized via sub-PlanningTables, are exposed to Case workers that are involved in planning.

Chapter 7 specifies semantics of run-time planning in detail, amongst others specifying when Stage instances become eligible for planning (into them) and until when planning can be performed. PlanningTables of HumanTasks, as well as the purpose of planning via HumanTasks, will be specified in 5.4.10.4.

PlanningTable inherits from CMMNElement, and has the following attributes:

Table 25: PlanningTable attributes

Attribute	Description
tableItems : TableItem[1..*]	A list of TableItem objects (see 5.4.9.1), available for planning. A PlanningTable is said to be "nested" in another PlanningTable, when the PlanningTable is a TableItem that is contained by that other PlanningTable, either directly, or recursively through even other PlanningTables. The set of tableItems of a PlanningTable MUST NOT include that PlanningTable or any PlanningTable in which that PlanningTable is nested. A PlanningTable MUST contain at least one TableItem.
applicabilityRules : ApplicabilityRule[0..*]	Zero or more ApplicabilityRule objects.

5.4.9.1 TableItem

A TableItem might be a DiscretionaryItem, or a PlanningTable.

TableItem inherits from CMMNElement and has the following attributes:

Table 26: TableItem attributes

Attribute	Description
authorizedRoleRefs : Role[0..*]	References to zero or more Role objects that are authorized to plan, based on the TableItem.
applicabilityRuleRefs : ApplicabilityRule[0..*]	References to zero or more ApplicabilityRule objects. If the condition of the ApplicabilityRule object evaluates to "true", then the TableItem is applicable for planning, otherwise it is not. If no ApplicabilityRule is associated with a TableItem, its applicability is considered "true". A PlanningTable that contains a TableItem MUST

	contain the ApplicabilityRules that represent the applicabilityRuleRefs of that TableItem.

5.4.9.2 DiscretionaryItem

A DiscretionaryItem identifies a PlanItemDefinition, of which instances can be planned, to the "discretion" of a Case worker that is involved in planning, which instances are planned into the context (see 5.4.9 and 7.6) that is implied by the PlanningTable that contains the DiscretionaryItem, either directly, or via a nested PlanningTable.

DiscretionaryItem inherits from TableItem and has the following attributes:

Table 27: DiscretionaryItem attributes

Attribute	Description
definitionRef : PlanItemDefinition[1]	Defines the PlanItemDefinition associated with the DiscretionaryItem, and which is the basis for planning. The definitionRef of a DiscretionaryItem MUST represent a Task or a PlanFragment (or Stage).
itemControl : PlanItemControl[0..1]	An optional PlanItemControl object. The PlanItemControl object controls aspects of the behavior of instances that are planned via the DiscretionaryItem. If the itemControl attribute is specified it MUST overwrite the value of attribute defaultControl of the DiscretionaryItem associated PlanItemDefinition

A PlanItemDefinition is said to be "discretionary" to a HumanTask or Stage, when the HumanTask or Stage has a PlanningTable, that, directly or through PlanningTable nesting, contains a DiscretionaryItem that refers to that PlanItemDefinition, or to a HumanTask or Stage, that has a PlanningTable, etc., ultimately arriving at a HumanTask or Stage that has a PlanningTable, that, directly or through PlanningTable nesting, contains a DiscretionaryItem that refers to that PlanItemDefinition.

A Stage MUST NOT be discretionary to itself or its nested Stages.

A Stage MUST NOT be discretionary to a HumanTask that is PlanItemDefinition of a PlanItem that is contained by the Stage or its nested Stages.

Unlike PlanItems, DiscretionaryItems do not have entryCriteriaRefs and exitCriteriaRefs in the model. The reason for that can be understood based on an example: Consider a PlanFragment, or Stage, that contains Tasks A and B, as PlanItems, whereby B is Sentry-based dependent on A. When the PlanFragment or Stage is planned, an instance of A and an instance of B go into the plan, in combination. The notion of Sentry-based dependency of that instance of B on that instance of A is going into the plan also. It is unambiguously defined which instance of B is dependent on which instance of A.

The situation with DiscretionaryItems is different. Consider a PlanningTable that contains Tasks C and D as DiscretionaryItems (C and D being discretionary Tasks). Multiple instances of C might be planned, as well as multiple instances of D, possibly at different moments, for different purposes, by different Case workers. When just the next instance of e.g. C is planned, the model cannot clarify on which instance(s) of D, already available in the plan, it would depend. An implementation might allow for Case worker interaction to establish such dependencies in the plan on the fly, but these dependencies do not come from the model (in design-time).

5.4.9.3 Applicability Rules

ApplicabilityRules are used to specify, whether a TableItem is “applicable” (“eligible”, “available”) for planning, based conditions that are evaluated over information in the CaseFile.

TableItems for which an associated ApplicabilityRule evaluates to “false”, will not be exposed to Case workers for planning purpose.

The class ApplicabilityRule has the following attributes:

Table 28: ApplicabilityRule attributes

Attribute	**Description**
name : String	The name of the ApplicabilityRule.
contextRef : CaseFileItem[0..1]	The context of the ApplicabilityRule. The caseFileItem that serves as starting point for evaluation of the Expression that is specified by the condition of the ApplicabilityRule. If not specified, evaluation starts at the CaseFile object that is referenced by the Case as its caseFileModel.
condition : Expression[1]	The Expression that serves as condition of the ApplicabilityRule. If it evaluates to “true”, then the associated TableItem is available for planning (if a Case worker is also assigned the Role that is authorized for planning based on that TableItem). Expressions are specified in 5.4.7.

5.4.10 Task

A Task is an atomic unit of work. Task is a base class for all Tasks in CMMN and inherits from PlanItemDefinition.

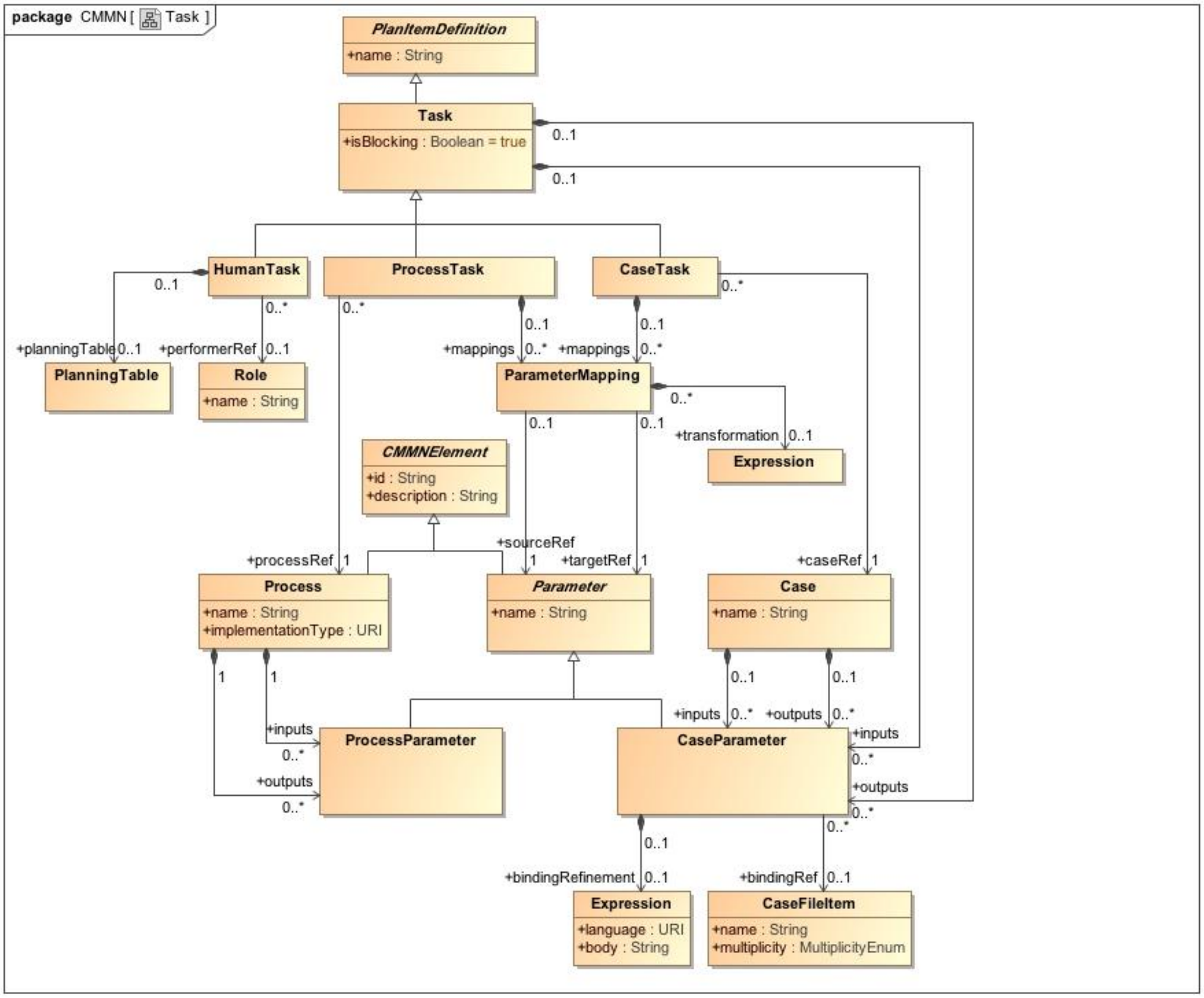


Figure 11: Task class diagram

The `Task` class has the following attributes:

Table 29: Task attributes and model associations

Attribute	**Description**
isBlocking : Boolean = true	If `isBlocking` is set to "true", the `Task` is waiting until the work associated with the `Task` is completed. If `isBlocking` is set to "false", the `Task` is not waiting for the work to complete and completes immediately, upon instantiation. The default value of attribute `isBlocking` MUST be "true". A `Task` that is non-blocking (`isBlocking` set to "false") MUST NOT have `outputs`.
inputs : CaseParameter[0..*]	Zero or more `CaseParameter` objects (see 5.4.10.3) that specify the `input` of the `Task`.

outputs : CaseParameter[0..*]	Zero or more CaseParameter objects (see 5.4.10.3) that specify the output of the Task.

5.4.10.1 Parameter

The class Parameter is an abstract base class for CaseParameter and ProcessParameter. It inherits from CMMNElelent, and has the following attributes:

Table 30: Parameter attributes

Attribute	Description
name : String	The name of the Parameter.

5.4.10.2 ParameterMapping

The class ParameterMapping is used for the input/output mapping of CaseTasks and ProcessTasks. It inherits from CMMNElement and has the following attributes:

Table 31: ParameterMapping attributes

Attribute	Description
transformation : Expression[0..1]	The transformation Expression transforms the parameter referred to by sourceRef to the parameter referred to by targetRef. Any expression language might be chosen for the transformation (for example XSLT, XPath, etc.) Expressions are specified in 5.4.7.
sourceRef : Parameter[1]	One source Parameter.
targetRef : Parameter[1]	One target Parameter.

5.4.10.3 CaseParameter

The class CaseParameter is used to model the inputs and outputs of Cases and Tasks.

It inherits from Parameter and has the following attributes:

Table 32: CaseParameter attributes

Attribute	Description

bindingRef : CaseFileItem[0..1]	A reference to a CaseFileItem. When a Task has an output that is a CaseParameter with bindingRef that references a CaseFileItem, the effect that the execution of instances of that Task has on instances of that CaseFileItem can be observed in terms of transitions in the CMMN-lifecyle of CaseFileItem (see 7.2). Similarly, when a Case has an input that is a CaseParameter with bindingRef that references a CaseFileItem, the effect that passing on information to an instance of the Case, via that CaseParameter, has on instances of that CaseFileItem in the CaseFile of that Case instance, can be observed in terms of transitions in the CMMN-lifecyle of CaseFileItem (see 7.2). Outputs of Cases and inputs of Tasks are merely concerned with retrieval of CaseFileItem (instances) from the CaseFile of a Case instance.
bindingRefinement : Expression[0..1]	An optional Expression to further refine the binding of the CaseParameter to the CaseFileItem, that it is referenced by the bindingRef of the CaseParameter. For example, if the bindingRef would refer to a CaseFileItem that represents a purchase order, the bindingRefinement might be used to effectively reduce the collection of referenced purchase orders to a particular purchase order (note that multiplicity of the CaseFileItem might be greater than zero), or to effectively refer to (an) associated CaseFileItem(s), such as (a) purchase order line(s). Expressions are specified in 5.4.7.

5.4.10.4 HumanTask

A HumanTask is a Task that is performed by a Case worker.

When a HumanTask is not “blocking” (isBlocking is “false”), it can be considered a “manual” Task, i.e. the Case management system is not tracking the lifecycle of the HumanTask (instance).

A HumanTask can have a PlanningTable, so that the HumanTask can also be used for planning Though planning can also be performed based on the PlanningTable of a Stage that contains the HumanTask, it has sometimes advantages to also perform planning from the HumanTask directly, such as:

- It brings a particular perspective of planning: TableItems in the PlanningTable of a HumanTask, that is used as PlanItem inside a Stage, are the basis for planning of Stages and Tasks that can be considered follow-up Stages and Tasks of that particular HumanTask. Planning based on the PlanningTable of the containing Stage, adds instances of Stages and Tasks that are contained by (an instance of) the Stage, but not particularly as follow-up of that HumanTask. The PlanningTable of the HumanTask typically contains TableItems that are particularly relevant in the context of planning from that particular HumanTask, whereas the PlanningTable of the containing Stage might provide a wider range of TableItems.
- It helps to avoid the overhead of defining “arbitrary” Stages that just contain a single PlanItem: In order to have a context with a more narrowly defined PlanningTable, it is often not preferred to

define further Stage nesting (by contained Stages that have their PlanningTables and that contain a HumanTask), but rather use a HumanTask with PlanningTable, which HumanTask is contained in the Stage directly.

- It allows to use the Role that is referenced by the performerRef of the HumanTask to effectively serve as the Role that is authorized to plan based on any TableItem in the PlanningTable of the HumanTask, or to enforce that Case workers that plan based on PlanItems in that PlanningTable have to be assigned both the HumanTask-related Role and the TableItem-related Roles.

HumanTask inherits from Task, and has the following attributes:

Table 33: HumanTask attributes

Attribute	**Description**
planningTable : PlanningTable[0..1]	An optional PlanningTable associated to the HumanTask. A HumanTask can be used for planning, and its PlanningTable might contain TableItems that are useful in the particular planning context. A HumanTask that is non-blocking (isBlocking set to “false”) MUST NOT have a PlanningTable.
performerRef : Role[0..1]	The performer of the HumanTask.

5.4.10.5 ProcessTask

A ProcessTask can be used in the Case to call a Business Process (see 5.4.10.5).

Parameters are used to pass information between the ProcessTask (in a Case) and the Process to which it refers: inputs of the ProcessTask are mapped to Inputs of the Process, and outputs of the ProcessTask are mapped to outputs of the Process. This way instances of (elements of) CaseFileItems from the CaseFile of the Case can be passed to the Process and outputs of the Process can be passed back and mapped to instances of (elements of) CaseFileItems.

When a ProcessTask is “blocking” (isBlocking is “true”), the ProcessTask is waiting until the Process associated with the ProcessTask is completed. If isBlocking is set to “false”, the ProcessTask is not waiting for the Process to complete, and completes immediately, upon its instantiation and calling its associated Process.

The class ProcessTask inherits from Task, and has the following attributes:

Table 34: ProcessTask attributes

Attribute	**Description**
processRef : Process[1]	A reference to a Process (see 5.4.10.5.1).
mappings : ParameterMapping[0..*]	Zero or more ParameterMapping objects. A ParameterMapping of a ProcessTask specifies how an input of the ProcessTask is mapped to an input of the called Process and how an output of the called Process is mapped to an output of the ProcessTask.

5.4.10.5.1 Process

A Process in CMMN is an abstraction of Processes as they are specified in various Process modeling specifications, in particular the ones that are listed in .

The class Process inherits from CMMNElement and has the following attributes:

Table 35: Process attributes

Attribute	Description
implementationType : URI	The implementation type of the Business Process. It MUST be provided in URI format
inputs : ProcessParameter[0..*]	Zero or more inputs of the Business Process
outputs : ProcessParameter[0..*]	Zero or more outputs of the Business Process

The following implementationTypes are defined to support various Business Process modeling standards:

Table 36: Process Implementation Types

Implementation Type URI	Description
http://www.omg.org/spec/CMMN/ProcessType/BPMN20	The Process to call is implemented in BPMN 2.0
http://www.omg.org/spec/CMMN/ProcessType/XPDL2	The Process to call is implemented in XPDL 2.x
http://www.omg.org/spec/CMMN/ProcessType/WSBPEL20	The Process to call is implemented in WS-BPEL 2.0
http://www.omg.org/spec/CMMN/ProcessType/WSBPEL1	The Process to call is implemented in WS-BPEL 1.x

5.4.10.6 CaseTask

A CaseTask can be used to call another Case. A CaseTask triggers the creation of an instance of that other Case, which creation denotes the initial transition in the CMMN-defined lifecycle of a Case instance (see 7.3).

The difference between using a CaseTask and a Stage is that a CaseTask calls a Case that has its own context, i.e. it is based on its own CaseFile, whereas a Stage represents behavior that shares the same context with the Stage, i.e. it is based on the same CaseFile and is “embedded” in the same Case.

Parameters are used to pass information between the CaseTask (in a Case) and the Case to which it refers: inputs of the CaseTask are mapped to Inputs of the Case, and outputs of the CaseTask are mapped to outputs of the Case. This way instances of (elements of) CaseFileItems can be exchanged between (CaseFiles of) Cases.

When a CaseTask is “blocking” (isBlocking is “true”), the CaseTask is waiting until the Case associated with the CaseTask is completed. If isBlocking is set to “false”, the CaseTask is not waiting for the Case to complete, and completes immediately, upon its instantiation and invocation of its associated Case.

The class CaseTask inherits from Task, and has the following attributes:

Table 37: CaseTask attributes

Attribute	Description
caseRef : Case[1]	A reference to the Case that is called as part of the CaseTask
mappings : ParameterMapping[0..*]	Zero or more ParameterMapping objects. A ParameterMapping of a CaseTask specifies how an input of the CaseTask is mapped to an input of the called Case and how an output of the called Case is mapped to an output of the CaseTask.

5.4.11 PlanItemControl

PlanItemControls define aspects of control of instances of Tasks, Stages, EventListeners and Milestones. They are defined in relation to their "origins" in the model - PlanItems and DiscretionaryItems - and maybe defaulted by PlanItemControls that are defined in relation to the PlanItemDefinitions to which the PlanItems and DiscretionaryItems refer to via their definitionRef.

PlanItemControls may specify the following:

- Under which conditions will Tasks and Stages, once enabled, start automatically. This is specified by AutomaticActivationRules, as part of PlanItemControls (see 5.4.11.1).
- Under which conditions will Tasks, Stages and Milestones be "required" to complete or terminate before their containing Stage can complete. This is specified by RequiredRules, as part of PlanItemControls (see 5.4.11.2)
- Under which conditions will Tasks, Stages and Milestones need to be repeated. This is specified by RepetitionRules, as part of PlanItemControls (see 5.4.11.3).

Run-time semantics in relation to these rules will be specified in chapter 7.

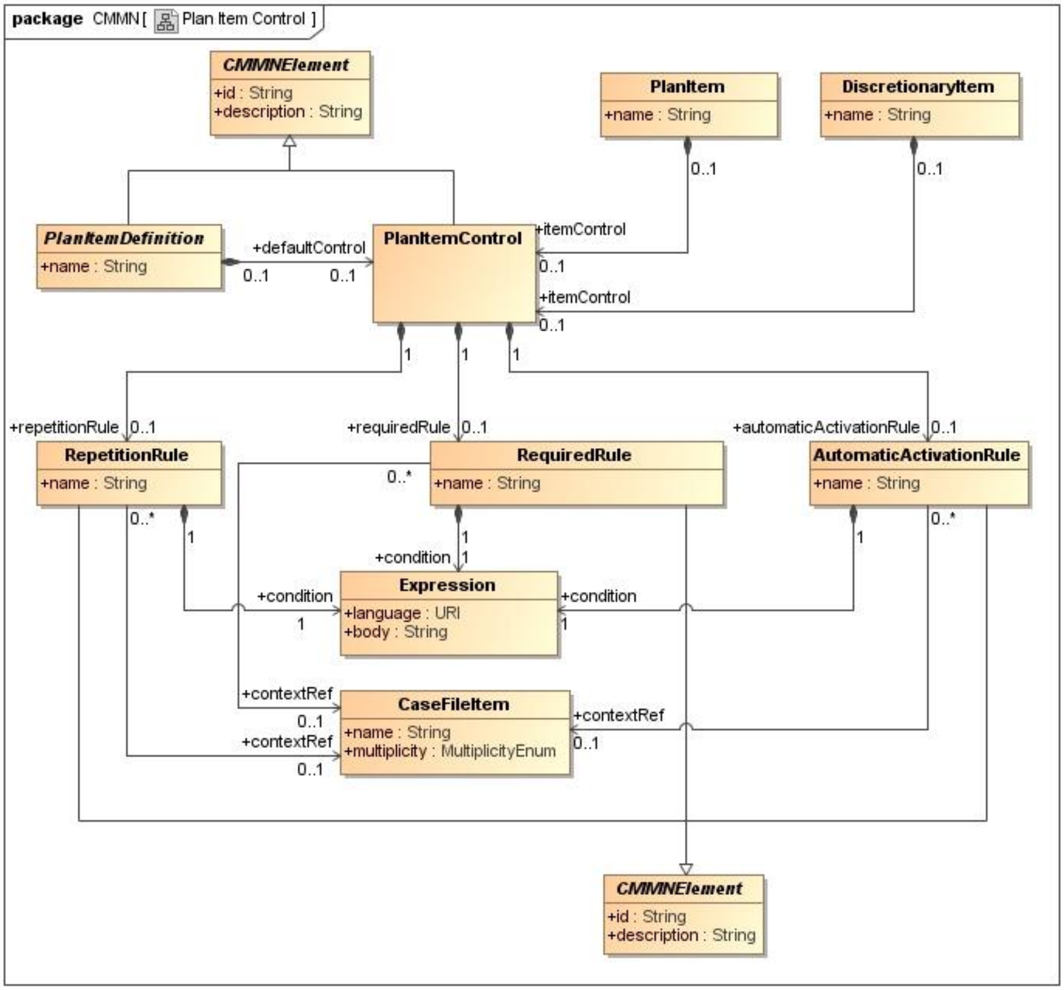


Figure 12: PlanItemControl class diagram

The class PlanItemControl inherits from CMMNElement and has the following attributes:

Table 38: PlanItemControl attributes and model associations

Attribute	Description
automaticActivationRule : AutomaticActivationRules[0..1]	Optional AutomaticActivationRule, as contained by the PlanItemControl. An AutomaticActivationRule comprises of an Expression that MUST evaluate to boolean. If no AutomaticActivationRule is specified then the default is considered “false”. A PlanItemControl that is the defaultControl of an EventListener or Milestone, or that is the itemControl of a PlanItem or DiscretionaryItem that is defined by an EventListener or Milestone, MUST NOT contain an AutomaticActivationRule.
requiredRule : RequiredRule[0..1]	Optional RequiredRule, as contained by the

	PlanItemControl. A RequiredRule comprises of an Expression that MUST evaluate to boolean. If no RequiredRule is specified, the default is “false”. A PlanItemControl that is the defaultControl of an EventListener, or that is the itemControl of a PlanItem or DiscretionaryItem that is defined by an EventListener, MUST NOT contain a RequiredRule.
repetitionRule : RepetitionRule[0..1]	Optional RepetitionRule, as contained by the PlanItemControl. A RepetitionRule comprises of an Expression that MUST evaluate to boolean. If no RepetitionRule object is specified, the default is “false”. A PlanItemControl that is the itemControl of a DiscretionaryItem, MUST NOT contain a RepetitionRule. (This is because the concept of “repetition” depends on the semantics of Sentries (see 5.4.11.3), and DiscretionaryItems are not associated with Sentries.) A PlanItemControl that is the defaultControl of an EventListener, or that is the itemControl of a PlanItem that is defined by an EventListener, MUST NOT contain a RepetitionRule. A PlanItem that has a PlanItemControl that contains a RepetitionRule, MUST have an entry criterion that refers to a Sentry that has at least one OnPart. (This is because the concept of “repetition” depends on the semantics of Sentries with onParts (see 5.4.11.3).)

A PlanItemControl MUST be the itemControl of a PlanItem or DiscretionaryItem or the defaultControl of a PlanItemDefinition.

A PlanItemControl MUST contain at least one repetitionRule or one requiredRule or one automaticActivationRule.

5.4.11.1 AutomaticActivationRule

An AutomaticActivationRule specifies under which conditions Tasks and Stages, once enabled, start automatically.

The class AutomaticActivationRule inherits from CMMNElement and has the following attributes:

Table 39: AutomaticActivationRule attributes

Attribute	Description
name : String	The name of the AutomaticActivationRule

contextRef : CaseFileItem[0..1]	The context of the AutomaticActivationRule. The caseFileItem that serves as starting point for evaluation of the Expression that is specified by the condition of the AutomaticActivationRule. If not specified, evaluation starts at the CaseFile object that is referenced by the Case as its caseFileModel.
condition : Expression[1]	A condition that is defined as Expression. Expressions are specified in 5.4.7. An Expression that MUST evaluate to boolean. If the expression evaluates to “true”, the instance of the Task or Stage MUST be activated automatically when it is in state Available, otherwise it MUST wait for manual activation (when it is in state Enabled) (see 7.3.2).

5.4.11.2 RequiredRule

A RequiredRule specifies under which conditions Tasks, Stages, EventListeners and Milestones will be “required” to complete or terminate before their containing Stage can complete.

The class RequiredRule inherits from CMMNElement and has the following attributes:

Table 40: RequiredRule attributes

Attribute	**Description**
name : String	The name of the RequiredRule
contextRef : CaseFileItem[0..1]	The context of the RequiredRule. The caseFileItem that serves as starting point for evaluation of the Expression that is specified by the condition of the RequiredRule. If not specified, evaluation starts at the CaseFile object that is referenced by the Case as its caseFileModel.
Condition : Expression[1]	A condition that is defined as Expression. Expressions are specified in 5.4.7. An Expression that MUST evaluate to boolean. If the Expression evaluates to “true”, then the instance of the Task, Stage, or Milestone is required and MUST be in state Disabled, Completed, Terminated, or Failed before its containing Stage (instance) can complete (see 7.3 and 7.5), otherwise it is considered optional.

5.4.11.3 RepetitionRule

A RepetitionRule specifies under which conditions Tasks, Stages and Milestones will have repetitions. Each repetition is a new instance of it. The trigger for the repetition is a Sentry, that is referenced as entry criterion, being satisfied, whereby an OnPart of that Sentry occurs. For example: A Task might be repeated each time a certain document is created. The Task (as PlanItem) might have an entry criterion, referring to a Sentry, having on OnPart, whereby the onPart refers to the CaseFileItem that represents the type of document, and whereby the standardEvent of the OnPart is specified as “create”. When the RepetitionRule as contained in the PlanItemControl of the Task (as PlanItem) also evaluates to “true”, the Task is repeated upon creation of the document.

EventListeners cannot have RepetitionRule. The notion of repetition is not useful for UserEventListeners. However, for a TimerEventListener repetition can be defined via a timerExpression based on ISO-8601, by defining repeating intervals in it (using "R<n>/" notation).

The class RepetitionRule inherits from CMMNElement and has the following attributes:

Table 41: RepetitionRule attributes

Attribute	**Description**
name : String	The name of the RepetitionRule
contextRef : CaseFileItem[0..1]	The context of the RepetitionRule. The caseFileItem that serves as starting point for evaluation of the Expression that is specified by the condition of the RepetitionRule. If not specified, evaluation starts at the CaseFile object that is referenced by the Case as its caseFileModel.
condition : Expression[1]	A condition that is defined as Expression. Expressions are specified in 5.4.7. An Expression that MUST evaluate to boolean. If the Expression evaluates to "true", then the instance of the Task, Stage, or Milestone maybe repeated, otherwise it MUST NOT be repeated.

The following table summarizes applicability of rules associated with PlanItemControl, in relation to Tasks, Stages, EventListeners and Milestones:

Table 42: Applicability of PlanItemControl rules

	RepetitionRule	**RequiredRule**	**AutomaticActivationRule**
Stage	Applicable	Applicable	Applicable
Task	Applicable	Applicable	Applicable
Milestone	Applicable	Applicable	N/A
EventListener	N/A	N/A	N/A

6 Notation

The following sections provide an overview of the CMMN notation used for modeling the core constructs of a Case.

6.1 Case

The CMMN notation provides for the depiction of the behavioral model elements of a Case (i.e. elements of a Case's casePlanModel). As far as modeling of information is concerned, only the information model elements (i.e. CaseFileItems) that are involved in the behavior of the Case are depicted. In other words, the CMMN notation does not provide for the visual modeling of the information model elements of the Case.

As with many other modeling languages, there are many different ways in which to model a Case using CMMN and its notation. It is left to the modeler to choose the best model to capture the essence of the situation at hand for the desired purpose.

6.2 Case Plan Models

The complete behavior model of a Case is captured in a casePlanModel. A casePlanModel is depicted using a "Folder" shape that consists of a rectangle with an upper left smaller rectangle attached to it. The name of the Case can be enclosed into the upper left rectangle.

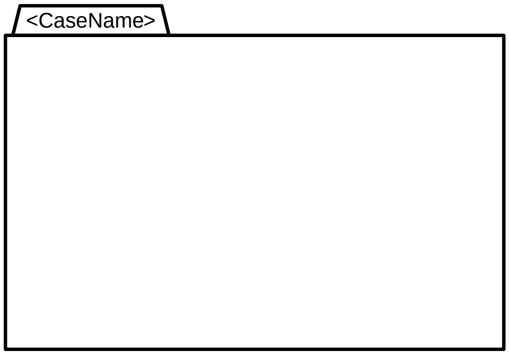


Figure 13: CasePlanModel Shape

The various elements of a casePlanModel are depicted within the boundary of the casePlanModel shape. Note that the casePlanModel is the outermost Stage that can be defined for a Case.

The following diagram shows an example of a Case's casePlanModel. Although incomplete, this diagram exemplifies the basis of Case modeling using the CMMN notation.

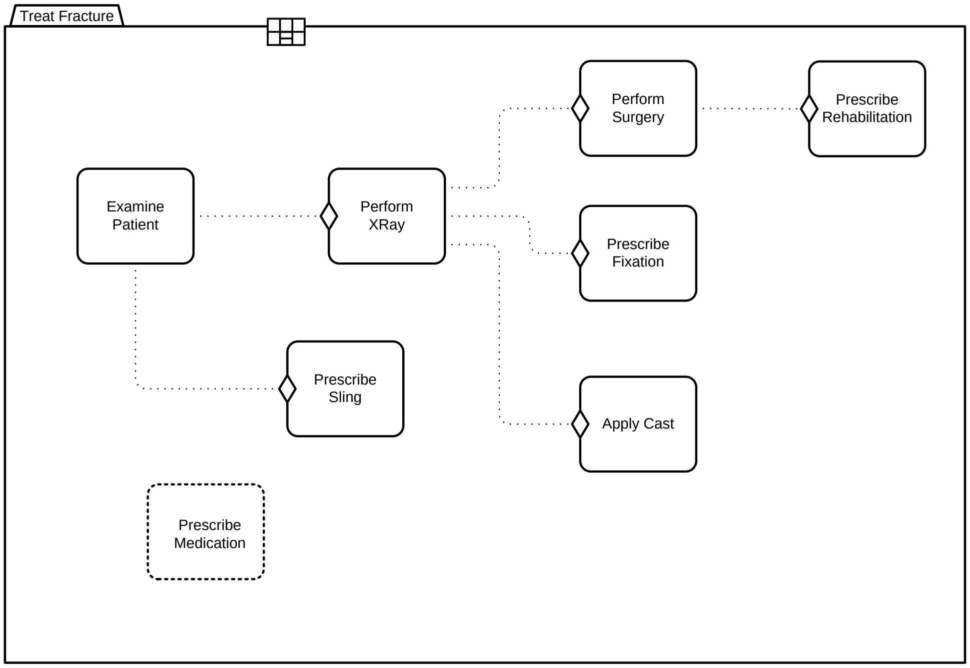


Figure 14: CasePlanModel Example

CMMN is declarative by nature, thus one should not read any meaning into the relative positioning of shapes.

6.3 Case File Items

A CaseFileItem is depicted by a “Document” shape that consists of a rectangle with a broken upper right corner.

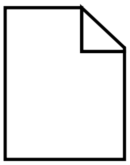


Figure 15: CaseFile**Item Shape**

6.4 Stages

A Stage is depicted by a rectangle shape with angled corners and a marker in the form of a “+” sign in a small box at its bottom center. When the Stage is expanded it is depicted by a rectangle shape with angled corners and a marker in the form of a “-” sign in a small box at its bottom center.

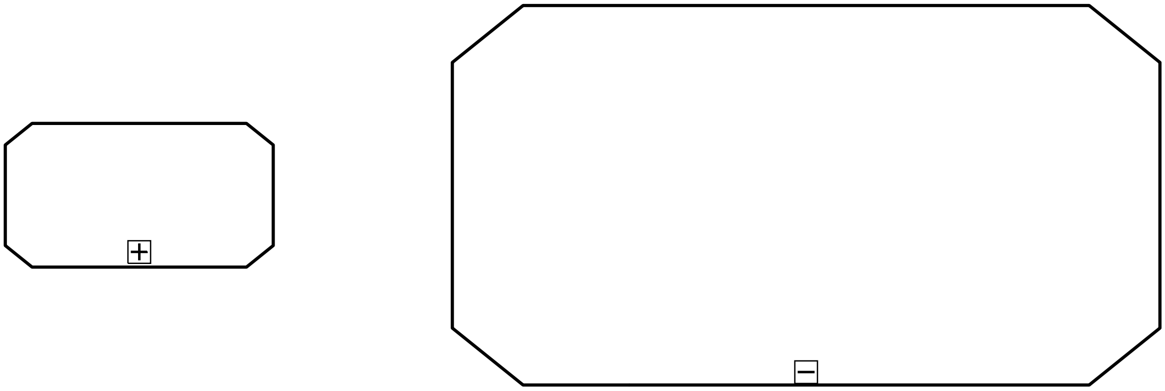


Figure 16: Collapsed Stage and Expanded Stage Shapes

A Stage may be discretionary (i.e used as DiscretionaryItem that is contained in a PlanningTable). A discretionary Stage has the shape of a rectangle with short dashed lines and angled corners and a marker in the form of a “+” sign in a small box at its bottom center, while a discretionary expanded Stage has the shape of a rectangle with short dashed lines and angled corners and a marker in the form of a “-” sign in a small box at its bottom center.

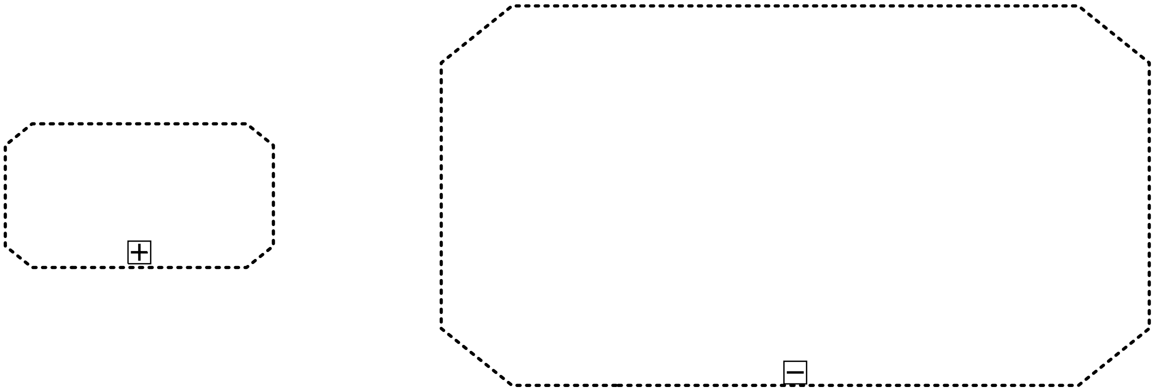


Figure 17: Discretionary Collapsed Stage and Discretionary Expanded Stage Shapes

When a `Stage` is expanded, elements that are contained in it become visible.

6.5 Entry and Exit Criterion

`PlanItem`s may have associated `Sentries`. When a `Sentry` is used as an entry criterion it is depicted by a shallow "Diamond" shape.

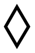


Figure 18: EntryCriterion Shape

When a `Sentry` is used as an exit criterion it is depicted by a solid "Diamond" shape.

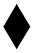


Figure 19: ExitCriterion Shape

When allowed, the Entry Criterion and Exit Criterion shapes can be placed as decorator anywhere on the boundary of a shape depicting the `PlanItem`.

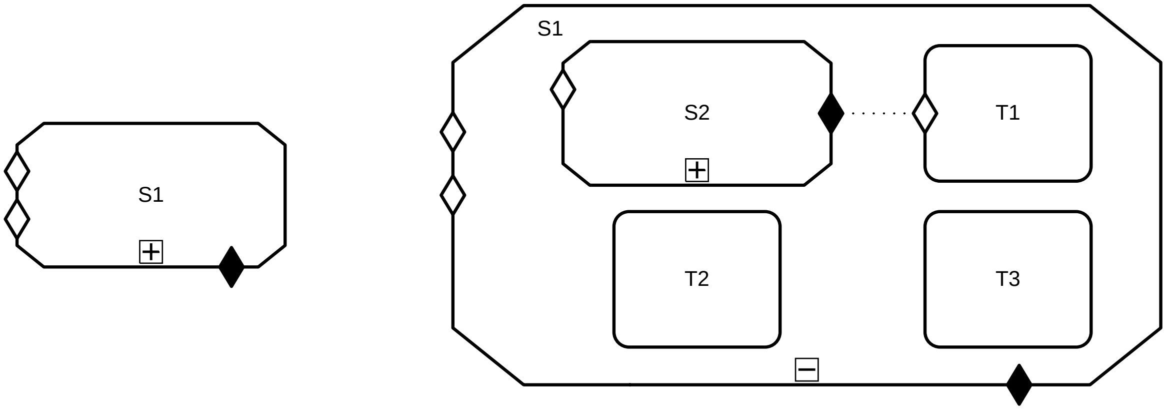


Figure 20: Collapsed and Expanded versions of a S**tage with two entry criterion, one sub** S**tage** and three T**asks**

6.6 Plan Fragments

A `PlanFragment` is depicted by a rectangle shape with dashed lines and softly rounded corners and a marker in the form of a “+” sign in small box at its bottom center. When the `PlanFragment` is expanded it is depicted by a rectangle shape with dashed lines and softly rounded corners and a marker in the form of a “-” sign in a small box at its bottom center.

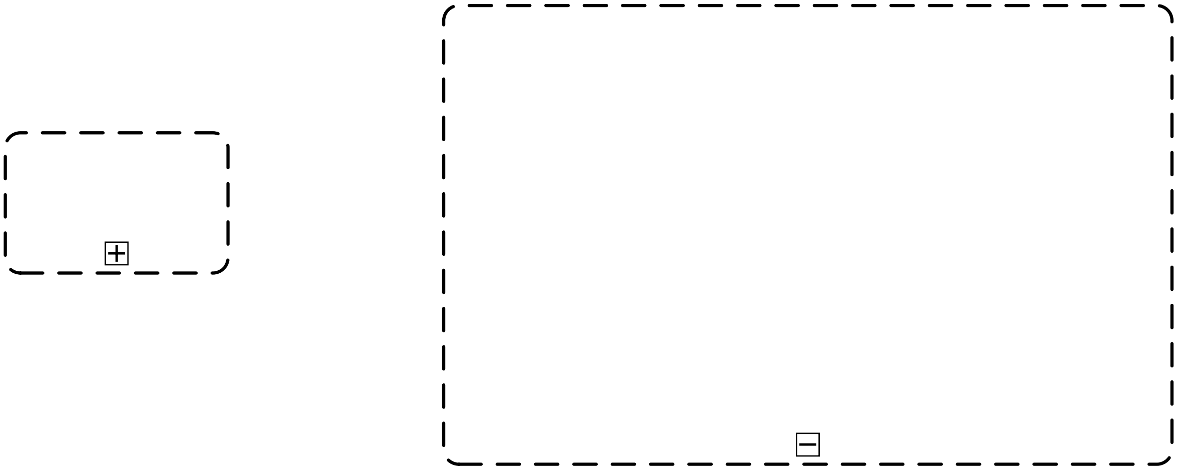


Figure 21: Collapsed PlanFragment and Expanded PlanFragment Shapes

When a `PlanFragment` is expanded, elements contained in it become visible.

6.7 Tasks

A `Task` is depicted by a rectangle shape with rounded corners.

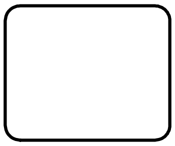


Figure 22: Task Shape

A `Task` may be discretionary (i.e. used as `DiscretionaryItem` contained in a `PlanningTable`). A discretionary `Task` is depicted by a rectangle shape with dashed lines and rounded corners

Figure 23: Discretionary Task

A `Task` may be associated with one or more entry criteria `Sentries` and one or more exit criteria `Sentries`.

The following example illustrates a `Task` with one entry criterion and one exit criterion.

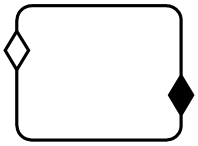


Figure 24: Task with one entry criterion and one exit criterion

6.7.1 Human Task

A `HumanTask` has two possible depictions. If the `HumanTask` is non-blocking (i.e. `isBlocking` set to "false"), it is depicted by a rectangle with rounded corners and a "Hand" symbol in the upper left corner. If the `HumanTask` is blocking (i.e. `isBlocking` set to "true"), it is depicted by a rectangle with rounded corners and a "User" symbol in the upper left corner.

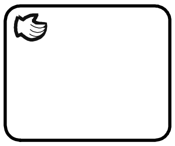


Figure 25: Non-blocking HumanTask Shape

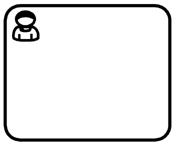


Figure 26: Blocking HumanTask Shape

A HumanTask may be discretionary (i.e. used as DiscretionaryItem contained in a PlanningTable). A discretionary HumanTask is depicted by a rectangle shape with dashed lines and rounded corners with the appropriate marker depending if it is blocking or not.

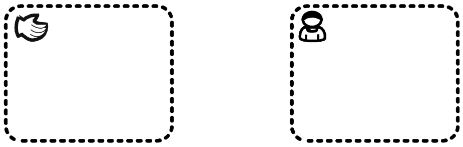


Figure 27: Non-Blocking and Blocking Discretionary HumanTasks

6.7.2 Case Task

A CaseTask is depicted by rectangle shape with rounded corners with a "Folder" symbol in the upper left corner.

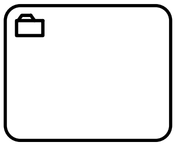


Figure 28: CaseTask Shape

A CaseTask may be discretionary (i.e. used as DiscretionaryItem containted in a PlanningTable). A discretionary CaseTask is depicted by a dash lined rectangle with rounded corners with a "Folder" symbol in the upper right corner.

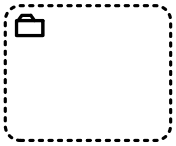


Figure 29: Discretionary CaseTask Shape

6.7.3 Process Task

A ProcessTask is depicted by a rectangle shape with rounded corners with a "Chevron" symbol in the upper left corner.

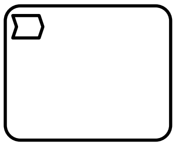


Figure 30: ProcessTask Shape

A ProcessTask may be discretionary (i.e. used as DiscretionaryItem contained in a PlanningTable). A discretionary ProcessTask is depicted by a dash lined rectangle with rounded corners with a “Chevron” symbol in the upper left corner.

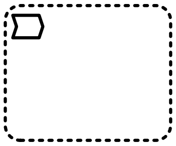


Figure 31: Discretionary ProcessTask Shape

6.8 Milestones

A Milestone is depicted by a rectangle shape with half-rounded ends.

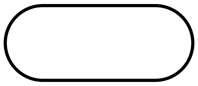


Figure 32: Milestone Shape

A Milestone may have zero or more entry criteria.

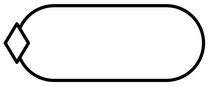


Figure 33: Milestone with one entry criterion

6.9 EventListeners

An EventListener is depicted by a double line circle shape with an open center so that markers can be placed within it to indicate variations of an EventListener. The circle MUST be drawn with a double line.

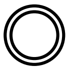


Figure 34: EventListener Shape

A TimerEventListener is depicted by double line circle shape with a “Clock” marker in the center.

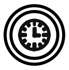


Figure 35: Timer**Event**Listener **Shape**

A UserEventListener is depicted by double line circle shape with a "User" symbol marker in the center.

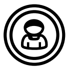


Figure 36: UserEventListener Shape

6.10 Connectors

Certain dependencies between elements that are shown inside expanded Stages or PlanFragments are depicted using connectors. The shape of the connector object is a dotted line. The connector MUST not have arrowheads.

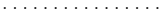


Figure 37: Connector Shape

One such depicted dependency is the onPart of a Sentry. For example, the following diagram illustrates a situation where the entry criteria of Task B depends on the completion of Task A.

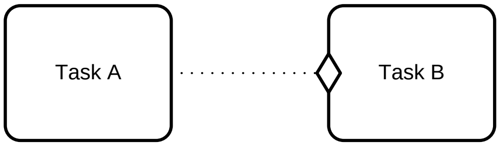


Figure 38: Sentry-based dependency between two Tasks

The other type of dependency that is visualized is the dependency between a HumanTask and DiscretionaryItems in its PlanningTable, when the HumanTask is shown with its PlanningTable expanded. These dependencies are also depicted by the same dotted line connector.

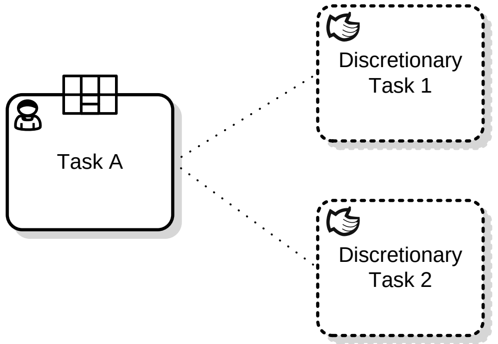


Figure 39: Dependency between a blocking HumanTask and its associated Discretionary Tasks

6.10.1 Connector Usage

Connectors that represent `Sentry` `onParts`, can be used to visualize (possibly complex) dependencies between `PlanItems`. The following picture illustrates a situation where `Task` C can be activated only if `Task` A and `Task` B complete.

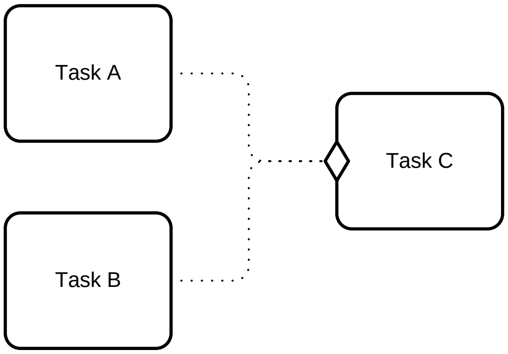


Figure 40: Using Sentry-based connectors to visualize "AND"

The following picture illustrates a situation where `Task` C can be activated if `Task` A or `Task` B completes.

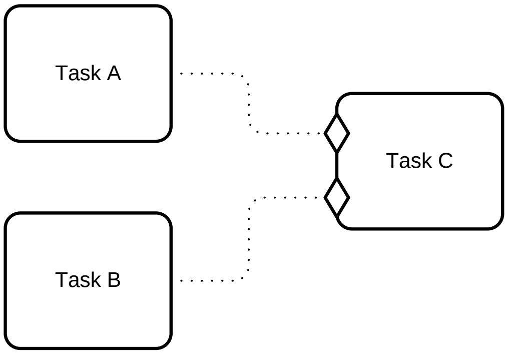


Figure 41: Using Sentry-based connectors to visualize "OR"

The following diagram illustrates a situation where Stage B depends on the exit criterion of Stage A.

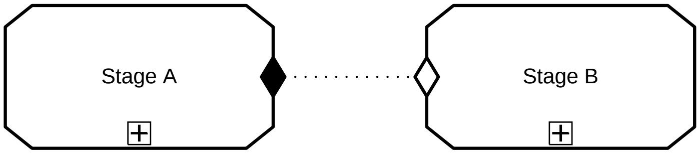


Figure 42: Using Sentry-based connector to visualize dependency between Stages

Note that the connection of the connector (i.e. onPart of the entry criterion Sentry of B) to the exit criterion Sentry of A visualizes the sentryRef of the onPart of the entry criterion Sentry of B (see 5.4.6.1).

The construct in Figure 43 maybe considered a "Stage transition", triggered by a particular event. Stage B is enabled via its entry criterion (depicted on its boundary), the OnPart of which may specify as standardEvent the termination of Stage A, given that it terminates based on the exit criterion (as depicted on its boundary). That exit criterion may itself has an OnPart (not depicted as connector) that refers e.g. to the creation of a document (CaseFileItem instance). So, when an instance of the document is created, Stage A terminates, and Stage B is enabled upon termination of Stage A, given that it terminates based on that document creation event.

The following diagram illustrates a situation where Task A depends on the achievement of Milestone A.

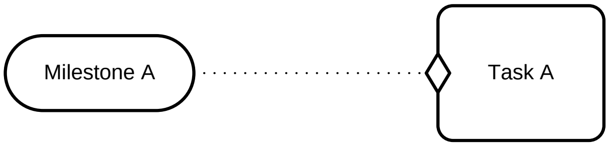


Figure 43: Using the Sentry-based connector to visualize dependency between a Task and a Milestone

The following diagram illustrates a situation where Task A depends on a TimerEventListener.

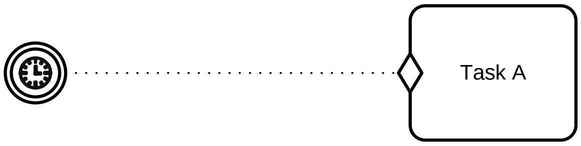


Figure 44: Using the Sentry-based connector to visualize dependency between a Task and a TimerEventListener

The following diagram illustrates a situation where Task A depends on a CaseFileItem.

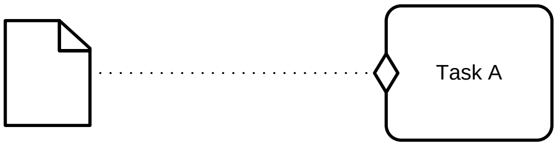


Figure 45: Using the Sentry-based connector to visualize dependency between a Task and a CaseFileItem

6.11 Planning Table

A Stage or a HumanTask can have a PlanningTable. A PlanningTable is depicted by a "Table" shape composed of six cells with the center bottom cell containing a marker indicating if the DiscretionaryItems are visualized or not. When DiscretionaryItems are NOT visualized a marker in the form of a "+" sign is present in the bottom center cell. When DiscretionaryItem are visualized a marker in the form of a "-" sign is present in the bottom center cell.

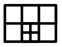


Figure 46: PlanningTable with DiscretionaryItems Not Visualized Shape

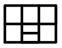


Figure 47: Planning Table with DiscretionaryItems Visualized Shape

The PlanningTable shape can only be placed as a decorator on the boundary of a Stage or a HumanTask object. The following example illustrates a Stage with a PlanningTable.

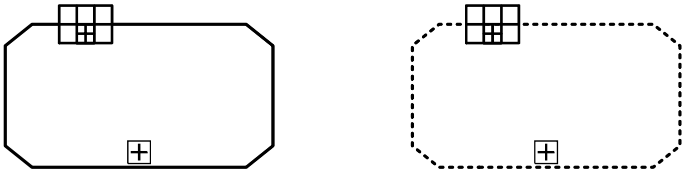


Figure 48: Stage and Discretionary Stage with PlanningTable

The following example illustrates a blocking HumanTask with a PlanningTable.

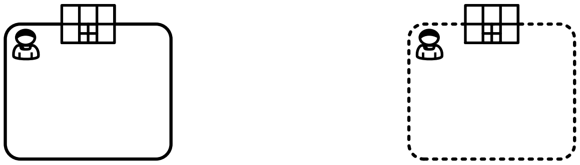


Figure 49: Blocking HumanTask and Discretionary Blocking HumanTask with PlanningTable

When a user "expands" a PlanningTable, its contained DiscretionaryItems become visible within the Stage.

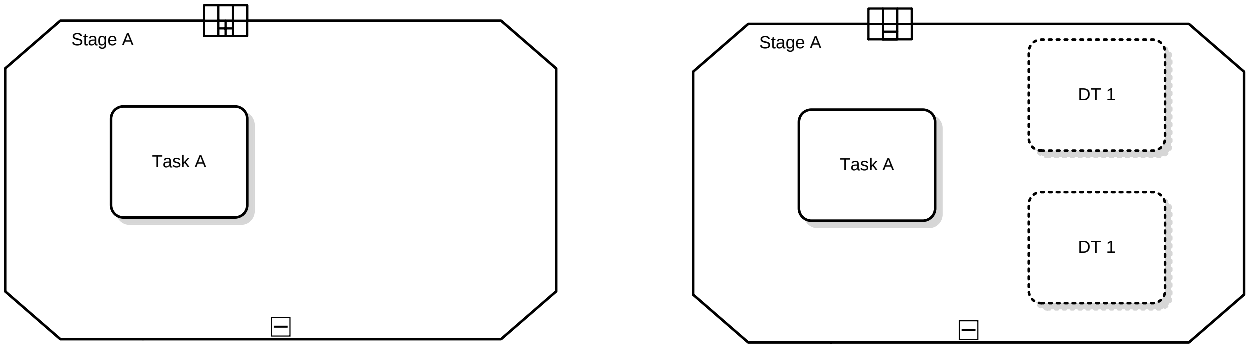


Figure 50: Stage with PlanningTable Collapsed and Expanded

When the `PlanningTable` of `HumanTask` is expanded, its contained `DiscretionaryItems` are visualized outside the `HumanTask` shape. The relationship between the `DiscretionaryItems` and the `HumanTask` is visualized with the dotted line connector.

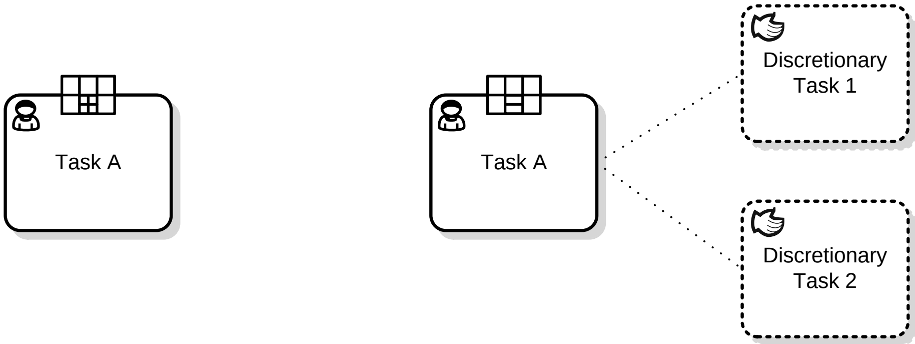


Figure 51: Blocking Human Task with DiscretionaryItems not expanded and expanded

The next four figures illustrate expansion of `PlanningTables`.

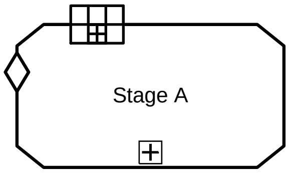


Figure 52: Collapsed Stage with Collapsed PlanningTable

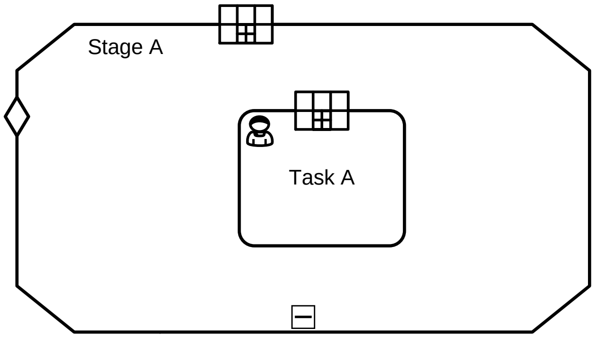


Figure 53: Expanded Stage with Collapsed PlanningTable

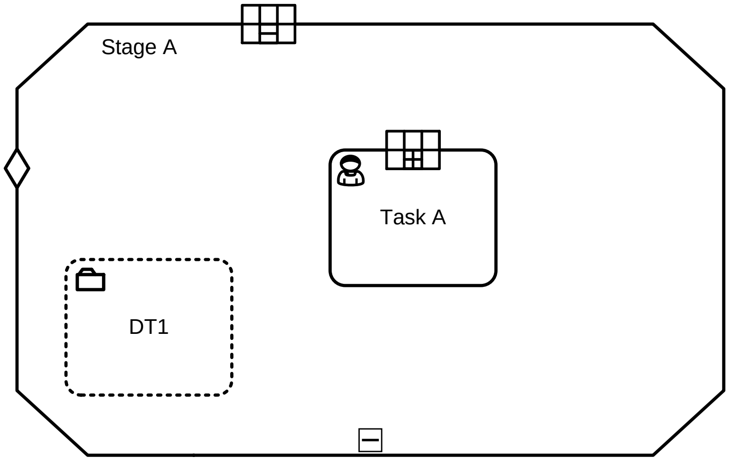


Figure 54: Expanded Stage with Expanded PlanningTable

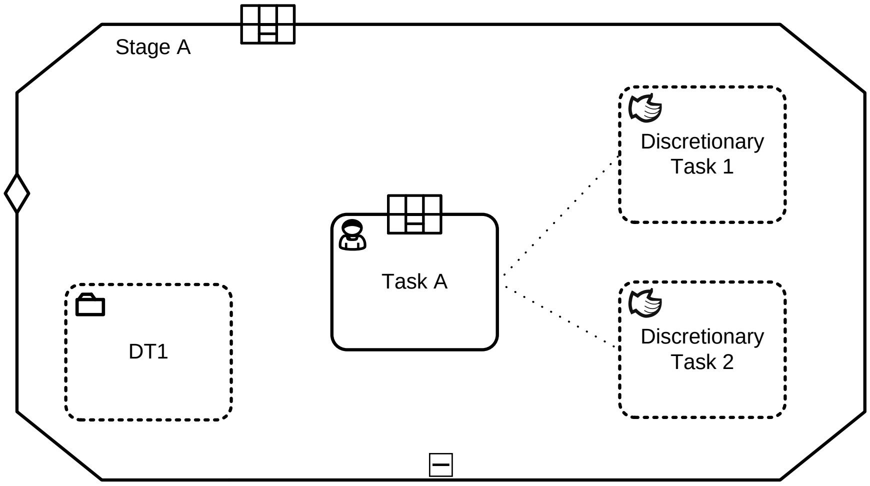


Figure 55: Expanded Stage with Expanded PlanningTable and Expanded HumanTask PlanningTable

6.12 Decorators

In order for the CMMN notation to be as expressive as possible, different shape decorators are introduced. These decorators are useful to visually indicate some particular behavior patterns of PlanItems and DiscretionaryItems.

6.12.1 AutoComplete Decorator

A when a Stage autoComplete attribute is set to “true”, then an AutoComplete decorator is added to the bottom center of the Stage shape.

The AutoComplete Decorator is a small black square.

Figure 56: AutoComplete Decorator

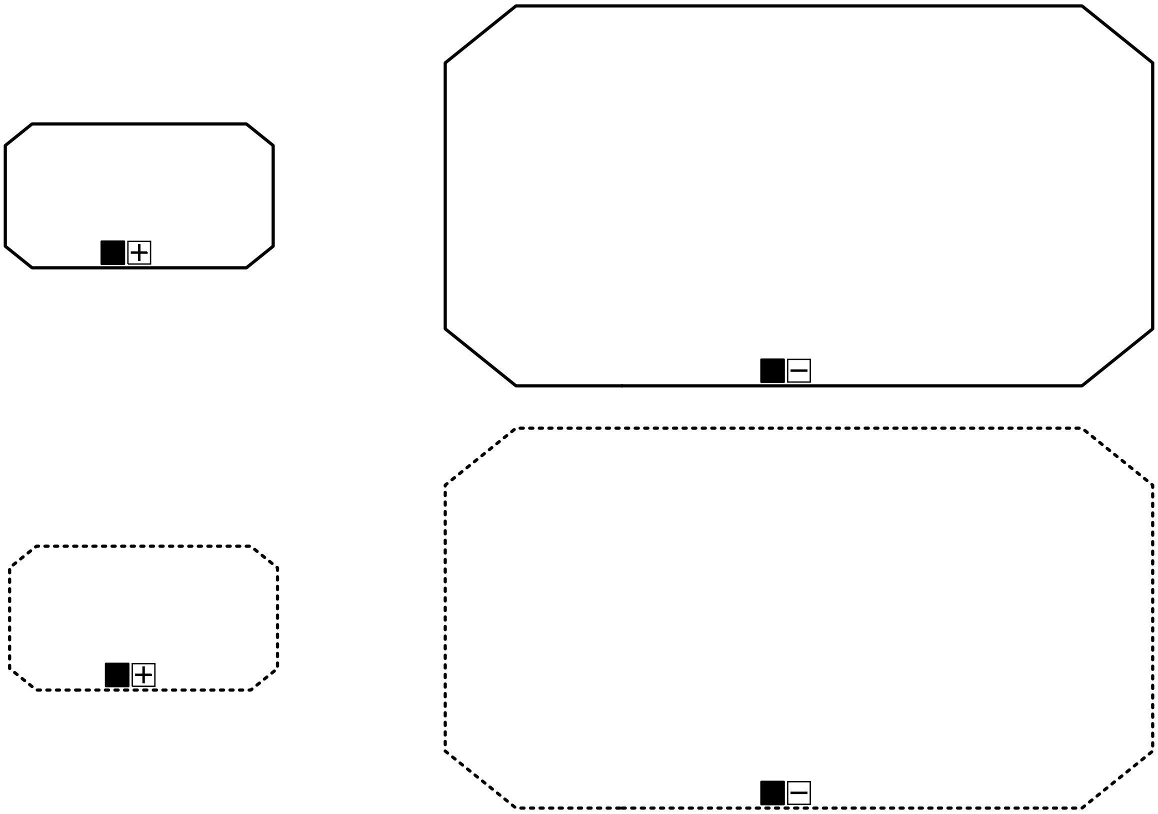


Figure 57: Stage Shape variations with AutoComplete Decorator

The next picture shows the outermost Stage of a Case, the casePlanModel, with AutoComplete Decorator.

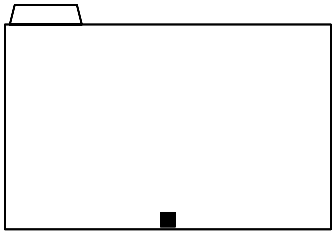


Figure 58: CasePlanModel with AutoComplete Decorator

6.12.2 AutomaticActivation Decorator

The AutomaticActivation Decorator, representing an `AutomaticActivationRule`, is a small black triangle pointing to the right.

Figure 59: AutomaticActivation Decorator

The automaticActivation Decorator is visible when an `AutomaticActivationRule` is defined for the `PlanItem` or `DiscretionaryItem`.

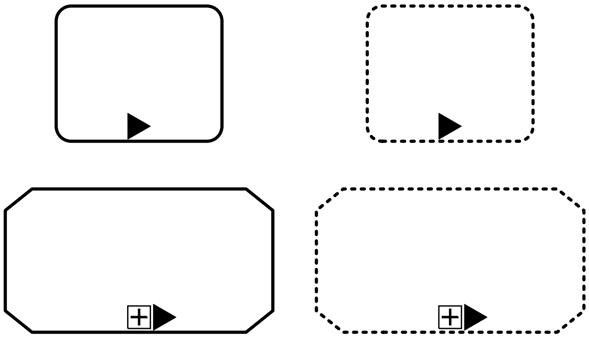


Figure 60: AutomaticActivation Decorator example on Task and Stage

6.12.3 Required Decorator

The Required Decorator is a bold black “Exclamation” symbol.

!

Figure 61: Required Decorator

The Required Decorator is visible when a `RequiredRule` is defined for `PlanItem` or `DiscretionaryItem`.

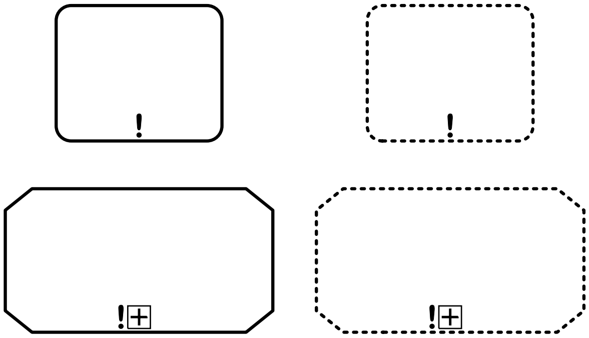


Figure 62: Required Decorator example on Task and Stage

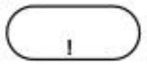


Figure 63: Required Decorator example on Milestone

6.12.4 Repetition Decorator

The Repetition Decorator, depicting a `RepetitionRule`, consists of three bold black bars.

Figure 64: Repetition Decorator

The Repetition Decorator is visible when a `RepetitionRule` is defined for a `PlanItem` or `DiscretionaryItem`.

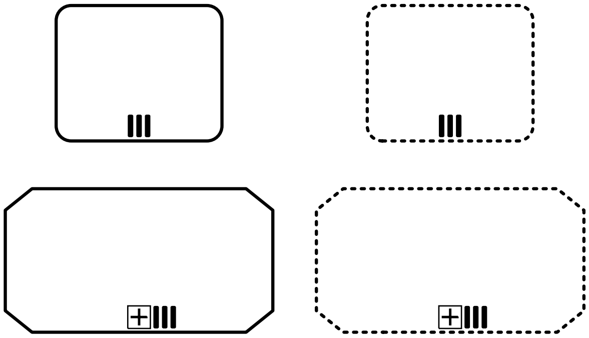


Figure 65: Repetition Decorator example on Task and Stage

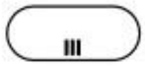
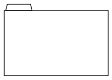
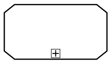
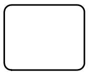
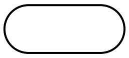
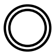
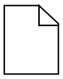
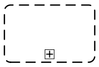


Figure 66: Repetition Decorator example on Milestone

6.12.5 Decorator Applicability Summary

Various Decorators can be added to CMMN shapes. The following table presents Decorators applicability.

Table 43: Decorators Applicability Summary Table

Decorator Applicability	Planning Table ⊞	Entry Critrion ◇	Exit Criterion ◆	AutoComplete ■	Automatic Activation ▶	Required !	Repetition ‖‖
CasePlanModel 	☑		☑	☑			
Stage 	☑	☑	☑	☑	☑	☑	☑
Task 	HumanTask only	☑	☑		☑	☑	☑
MileStone 		☑				☑	☑
EventListener 							
CaseFileItem 							
PlanFragment 							

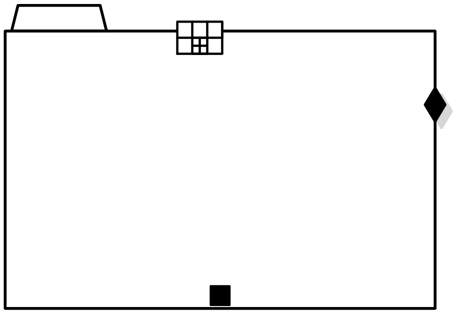


Figure 67: CasePlanModel Shape with all possible Decorators

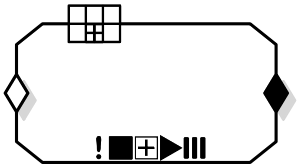


Figure 68: Stage Shape with all possible Decorators

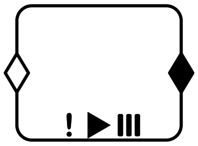


Figure 69: Task Shape with all possible Decorators

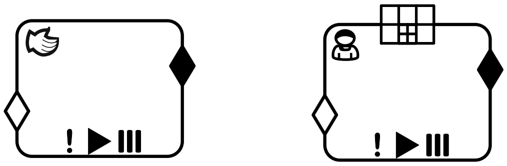


Figure 70: Non-Blocking and Blocking HumanTask Shapes with all possible Decorators

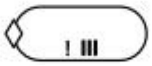


Figure 71: Milestone Shape with all possible Decorators

6.13 Examples

The following picture shows a combination of various elements, by means of a small example, which is about writing a document.

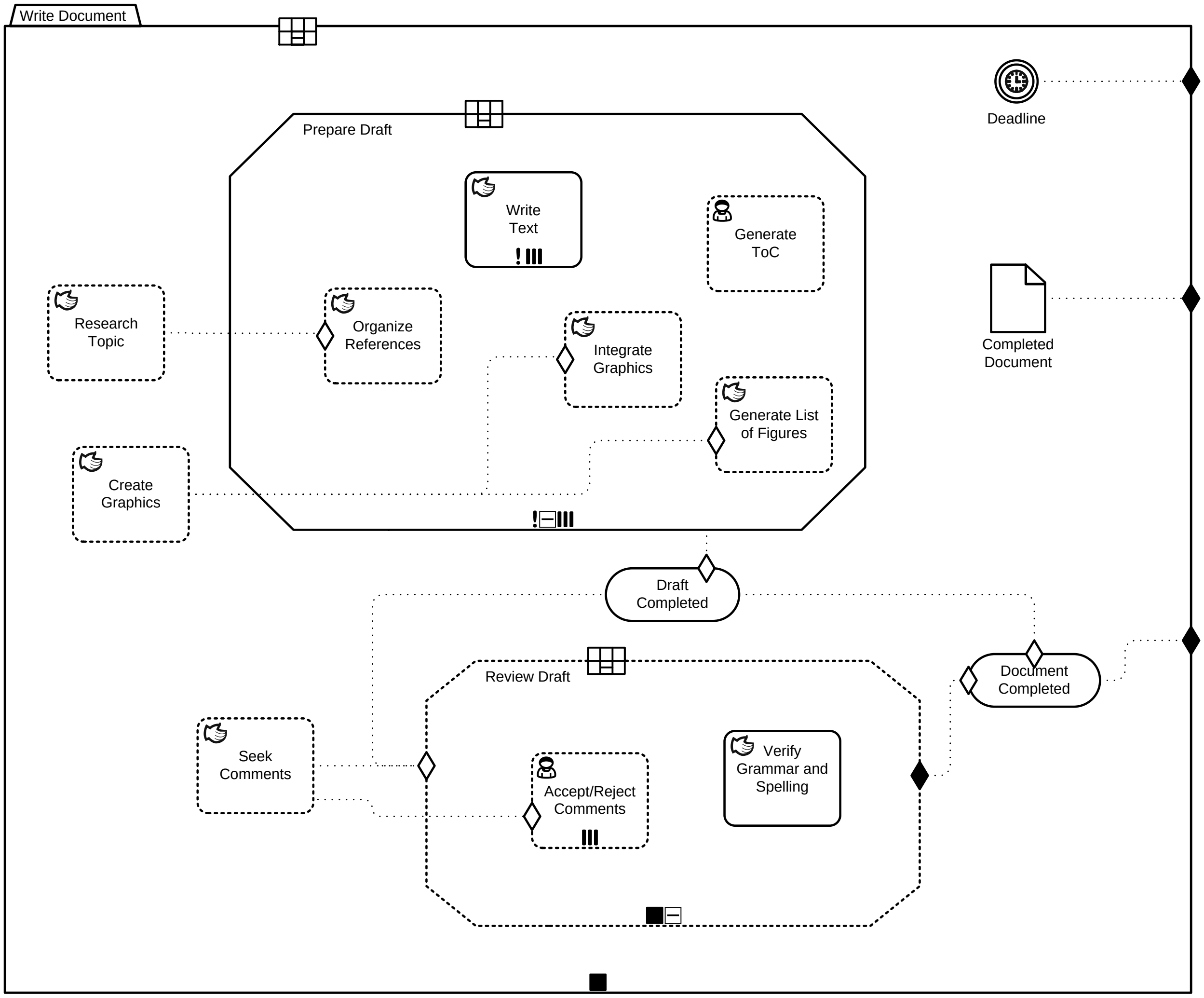


Figure 72: Write Document Example

7 Execution Semantics

Most of the execution semantics is described by the lifecycle of important CMMNElement instances. In particular the lifecycle for Task, Stage, Milestone, EventListener, and CaseFileItem instances describe the majority of the execution semantics. In addition to the lifecycle there are behavioral property rules that also describe the behavior of a case management system.

This chapter first describes the overall semantics associated with Case instances. It then describes the semantics of the caseFileModel, and concludes with the semantics of the casePlanModel portion.

7.1 Case Instance

A Case instance is composed of information represented by a caseFileModel and behavior represented by a casePlanModel. In addition, there are roles, which correspond to humans expected to participate in the Case.

When a Case instance is created, the caseFileModel, casePlanModel, and caseRoles are all initialized. The Stage instance implementing the casePlanModel starts executing in an Active state (see 7.3), and while the Case instance is not in Closed state the caseFileModel can be modified, planning can occur, and human participants can be assigned to roles.

7.2 CaseFileItem Lifecycle

The following diagram illustrates the lifecycle of a CaseFileItem instance:

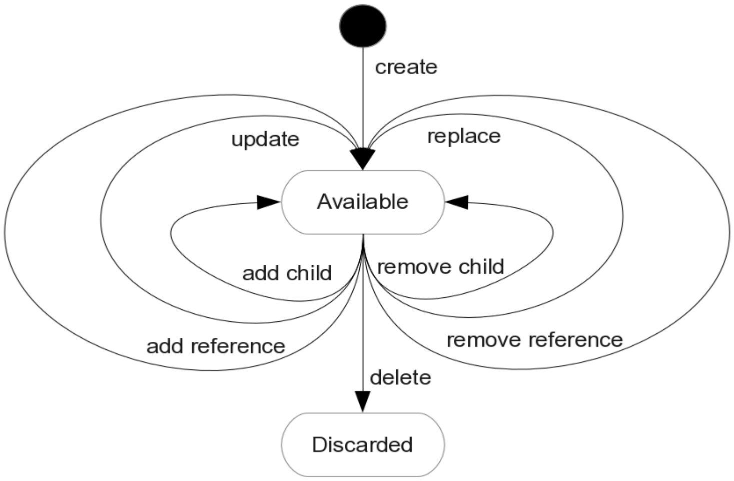


Figure 73: CaseFileItem instance lifecycle

A CaseFileItem instance has the following states:

Table 44: CaseFileItem instance states

State	Description
Available	In this state a CaseFileItem instance is available for Case workers to use
Discarded	A CaseFileItem instance in this state is considered deleted and is not available to Case workers or expressions

A CaseFileItem instance can undergo the following transitions:

Table 45: CaseFileItem instance transitions

Transition	From	To	Description
create	Ø	Available	Transition to the Available state when a CaseFileItem instance is created
update	Available	Available	Transition when a CaseFileItem instance property is updated
replace	Available	Available	Transition when the CaseFileItem instance content is replaced
add child	Available	Available	Transition when another CaseFileItem instance is added to the children relationship (meaning an entry is added to CaseFileItem.children)
remove child	Available	Available	Transition when another CaseFileItem instance is removed from the children relationship (meaning an entry is removed from CaseFileItem.children)
add reference	Available	Available	Transition when another CaseFileItem instance is added to the target reference relationship (meaning an entry is added to CaseFileItem.targetRef)
remove reference	Available	Available	Transition when another CaseFileItem instance is removed from the targetRef relationship (meaning an entry is removed from CaseFileItem.targetRef)
delete	Available	Discarded	Terminal state

A Task instance output MAY have an effect on CaseFileItem instances that are specified as output CaseParameters of a Task instance.

7.2.1 CaseFileItem operations

The following standard operations are defined for CaseFileItem instances to support navigation over the CaseFile:

Table 46: CaseFileItem instance operations

Operation	Parameters	Description
getCaseFileItemInstance	**IN** itemName : String **OUT** CaseFileItem instance	Get a CaseFileItem instance of given itemName. If no CaseFileItem instance for the given itemName exists, an empty CaseFileItem instance MUST be returned.
getCaseFileItemInstance	**IN** itemName : String	Get a CaseFileItem instance of given itemName and index. This operation MUST be used for CaseFileItem

	index : Integer **OUT** CaseFileItem instance	instances with a multiplicity greater than one. The index is used to identify a concrete CaseFileItem instance from the collection of CaseFileItem instances. If no CaseFileItem instance for the given itemName exists, or if the index is out of the range of CaseFileItem instances, an empty CaseFileItem instance MUST be returned.
getCaseFileItemInstanceProperty	**IN** item : CaseFileItem instance propertyName : String **OUT** Element	Get the value of a CaseFileItem instance property. If propertyName refers to a non-existing property of the CaseFileItem instance, an empty Element MUST be returned. The Element returned MUST be of the specified property type for the CaseFileItem instance.
getCaseFileItemInstanceChild	**IN** item : CaseFileItem instance childName : String **OUT** CaseFileItem	Get a child CaseFileItem instance for a given CaseFileItem instance. The value of parameter childName specifies the name of the child to get. If no child of the given name exists for the CaseFileItem instance, an empty CaseFileItem instance MUST be returned.
getCaseFileItemInstanceParent	**IN** item : CaseFileItem instance **OUT** CaseFileItem instance	Get the parent CaseFileItem instance of a CaseFileItem instance. If no parent exists then an empty CaseFileItem instance MUST be returned.
getCaseFileItemInstanceTarget	**IN** item : CaseFileItem instance targetName : String **OUT** CaseFileItem instance	Get a target CaseFileItem instance for a given CaseFileItem instance. The value of parameter targetName specifies the name of the target to get. If no target of the given name exists for the CaseFileItem instance, an empty CaseFileItem instance MUST be returned.
getCaseFileItemInstanceSource	**IN** item : CaseFileItem instance **OUT** CaseFileItem instance	Get the source CaseFileItem instance of a CaseFileItem instance. If no source exists then an empty CaseFileItem instance MUST be returned.

An implementation that uses XPath as expression language (see 5.1.2), MIGHT use XPath Extension Functions to implement those operations.

7.3 CasePlanModel Lifecycles

The behavior associated with Case models in CMMN is the result of combining a variation of the operational semantics for business artifacts managed based on the guard-stage-milestone (GSM) concept, with other concepts, such as, most notably, dynamic planning, the application of finite state machine lifecycles for CaseFileItem, EventListener, Milestone, Stage and Task instances, and application of so-called Behavior Property Rules (see 7.5). Further generalizations include the possibility that PlanItemDefinitions may have multiple, simultaneous occurrences, and the separation of Milestones from Stages (in GSM, each milestone is associated with a stage, and achieving the milestone has the effect of terminating the stage).

Stages contain other PlanItems (Stages, Tasks, Milestones, and EventListeners). The terminology used in this specification, calls the elements inside a Stage its children, and the container Stage the parent. Therefore, a child of a Stage is an element contained in that Stage. The parent of an element is the Stage that contains that element. This terminology refers to a single level of containment; a second level of containment may be referenced using grandchildren or grandparent. For example for a Stage S1 containing a single Stage S2 that itself contains a single Task T1, this specification will say that S1 is the parent of S2, and S2 is the parent of T1, T1 is the only child of S2, and S2 is the only child of S1.

This section describes the lifecycle of some important CMMNElement instances, including Case and all the PlanItemDefinition derived classes (Stage, Task, Milestone, and EventListener) instances.

It is important to understand that when we talk about EventListener, Milestone, Stage or Task instances we refer to the instances that originate from instantiating a PlanItemDefinition that is referred from a PlanItem or DiscretionaryItem associated with the corresponding EventListener, Milestone, Stage or Task.

There are nine states used in these lifecycles, and they are described in the following table.

Table 47: Case, EventListener, Milestone, Stage and Task instance states

State	Description
Active	Indicates behavior is being executed in the instance
Available	The instance is waiting for a Sentry to become “true” or for an event to occur, so that the instance can progress to its primary purpose (e.g., become Active or Enabled)
Closed	Terminal state. There is no activity (no behavior being executed) in the Case instance, and further planning in the Case's casePlanModel is not permitted. This state is only available for the outermost Stage instance implementing the Case's casePlanModel
Completed	Semi-terminal state[1] for Case instance, but terminal state for all other EventListener, Milestone, Stage or Task instances. There is no activity (no behavior being executed) in the element. A Case instance could transition back to Active by engaging in planning at the outermost Stage instance implementing the Case's casePlanModel

[1] For the purpose of this specification, a semi-terminal state is a state with a transition out of the state, but it is considered terminal to calculate Completion state of its parent Stage instance.

State	Description
Disabled	Semi-terminal state. Indicates a `Case` worker (human) decision to disable the instance, because it may not be required for the `Case` instance at hand
Enabled	The instance is waiting for a `Case` worker (human) decision to become Active or Disabled
Failed	Semi-terminal state. This state indicates an exception or software failure.
Suspended	Indicates a `Case` worker (human) decision to temporary suspend work on an Active instance. There is no activity (no behavior being executed) in the instance, but a `Case` worker (human) could move the instance back to an Active state.
Terminated	Terminal state. Indicates termination by an exit criteria or a `Case` worker (human) decision to terminate an Active instance.

Terminal states (Closed, Completed, and Terminated) and semi-terminal states (Disabled, and Failed) are used to calculate the completion of its enclosing `Stage` instance. A semi-terminal state is a state with a transition out of the state, but it is considered terminal to calculate Completion state of its parent `Stage` instance.

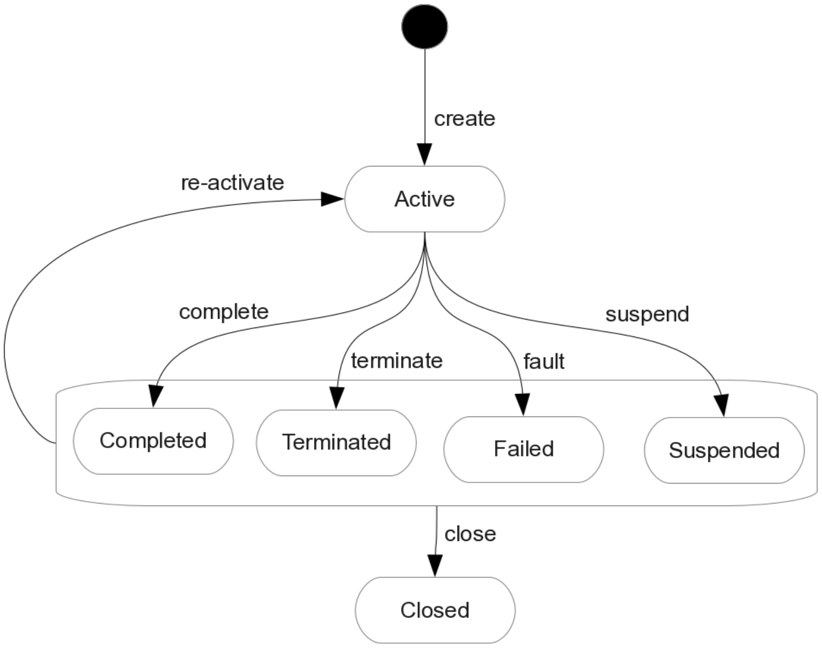
7.3.1 Case Instance Lifecyle

The `Case` lifecycle corresponds to the `Stage` instance implementing the `Case’s casePlanModel`, which in below text is referred as the outermost `Stage` instance of the `Case` instance. The outermost `Stage` instance is special in two areas:

1- It MUST NOT contain entry criteria.

2- That `Stage` instance implements the `Case` lifecycle described in this section, which is different than the lifecycle for all other `Stage` instances.

The following diagram illustrates the lifecycle of a `Case` instance, by illustrating the lifecycle of the `Case's casePlanModel`.



(Figure 74: Lifecycle of a Case instance

A Case instance has the following states:

Table 48: Case instance states

State	Description
Active	In this state the Case instance is executing; meaning the outermost Stage instance is in the Active state
Suspended	This state allows a Case worker (human) to temporarily suspend an executing Case instance. A Case instance MUST propagate this state to its outermost Stage instance. This state MUST then be propagated down to the outermost Stage instance's contained EventListener, Milestone, Stage and Task instances.
Completed	The Case instance is completed, when all the required Milestone, Stage and Task instances in the outermost Stage instance are completed (completed or terminated), and there are no executing (Active) Stage or Task instances.
Terminated	Terminal state. This state can be achieved by an exit criteria and also allows a Case worker (human) to terminate an executing Case instance. This state is reached when the outermost Stage instance reaches it.
Failed	Semi-terminal state. This state is reached when the outermost Stage instance reaches it. The state indicates an exception or software failure.
Closed	Terminal state. In this state no new activity is allowed in the Case. The Case instance caseFileModel and all its content becomes read only, and no new Task or Stage instances can be planned

A Case instance can undergo the following transitions:

Table 49: Case instance transitions

Transition	From	To	Description
create	Ø	Active	Transition to the initial state (Active) when the Case instance is created. The outermost Stage instance skips the Available state and MUST transition directly to the Active state, because that Stage instance does not have a (entry criteria) Sentry.
suspend	Active	Suspended	Transition by Case worker (human) decision. This state propagates down to the outermost Stage instance, which in turn propagates it down to all its internal EventListener, Milestone, Stage and Task instances.
terminate	Active	Terminated	Transition by Case worker (human) decision. This state propagates down to the outermost Stage instance, which in turn propagates it down to all its internal EventListener, Milestone, Stage and Task instances
complete	Active	Completed	Transition when all the required Milestone, Stage and Task instances have reached a terminal state (Closed, and Terminated) or a semi-terminal state (Completed, Disabled, and Failed), and there are no executing (Active) Stage or Task instances.
fault	Active	Failed	Transition when the outermost Stage instance reaches the Failed

Transition	From	To	Description
			state due to an exception or software failure
re-activate	Completed Terminated Failed Suspended	Active	Transition by a Case worker (human), or an administrator.
close	Completed Terminated Failed Suspended	Closed	Transition by the system, an administrator, or Case worker (human) when no further work or modifications should be allow for this Case

7.3.2 Stage and Task Lifecycle

The following diagram illustrates the lifecycle of a Stage or Task instance:

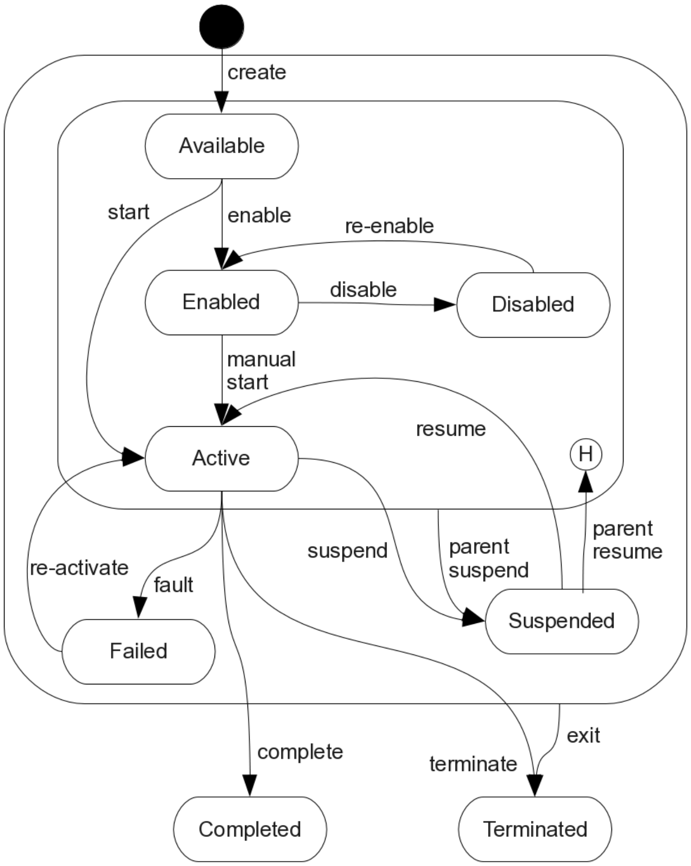


Figure 75: Lifecycle of a Stage or Task instance

A Stage or Task instance has the following states:

Table 50: Stage and Task instances states

State	Description
Available	A Stage or Task instance becomes available when the Stage instance in which it resides moves into Active state. While available, the Stage or Task instance is waiting for its entry criteria (Sentry) to become "true". A missing entry criteria (Sentry) is considered "true".
Enabled	A Stage or Task instance in this state is waiting for a human to start or disable it. Only Stage or Task instances that require Case worker (human) intervention to start get into this state (AutomaticActivationRule evaluates to "false")
Disabled	Semi-terminal state. This state is reached when a Case worker (human) decides the Stage or Task instance should not execute in this instance of the Case
Active	The Stage or Task instance is executing in this state. Stage instances in this state contain: • At least one Stage or Task instance in the Available, Enabled, Active, or Suspended state, and/or • No Stage or Task instance at all but an associated PlanningTable and for a Stage instance, autoComplete is set to "false".
Suspended	This state allows a Case worker (human) to temporarily suspend an executing Stage or Task instance. A Stage instance MUST propagate this state to all its contained EventListener, Milestone, Stage and Task instances.
Failed	Semi-terminal state. This state indicates an exception or software failure
Completed	Terminal state. This state indicates normal termination of the Stage or Task instance. For a Stage instance it indicates all its contained Stage or Task instances MUST be either completed or terminated.
Terminated	Terminal state. This state indicates a termination by a Case worker (human), or termination by reaching the exit criteria sentry. A Stage instance MUST propagate this state to all its contained EventListener, Milestone, Stage and Task instances.

A Stage or Task instance can undergo the following transitions:

Table 51: Stage and Task instance transitions

Transition	From	To	Description
create	Ø	Available	Transition to the initial state (Available) when the Stage or Task instance is created. This happens when the Stage instance containing this Stage or Task instance transitions to Active. The RepetitionRule and the RequiredRule Boolean expressions MUST be evaluated in this transition, and their Boolean values SHOULD be maintained for the rest of the life of the Stage or Task instance.
enable	Available	Enabled	Transition when the entry criteria (sentry) becomes "true" and the Stage or Task instance requires manual intervention to transition to Active or Disabled. This transition only happens if the AutomaticActivationRule evaluates to "false"

Transition	From	To	Description
			at the moment the sentry becomes "true". The AutomaticActivationRule Boolean expression MUST be evaluated in this transition and its Boolean value SHOULD be maintained for the rest of the life of the Stage or Task instance.
start	Available	Active	Transition when the entry criteria (sentry) becomes "true" and the Stage or Task instance does not require manual intervention. This transition only happens if the AutomaticActivationRule evaluates to "true" at the moment the sentry becomes "true". The AutomaticActivationRule Boolean expression MUST be evaluated in this transition, and its Boolean value SHOULD be maintained for the rest of the life of the Stage or Task instance.
disable	Enabled	Disabled	Transition by Case worker (human) decision
manual start	Enabled	Active	Transition by Case worker (human) decision
suspend	Active	Suspended	Transition by Case worker (human) decision or propagation from outer Stage instance. For a Stage instance, this state MUST propagate to all its contained EventListener, Milestone, Stage and Task instances.
fault	Active	Failed	Transition when an exception or software failure occurs. This state MUST NOT propagate.
complete	Active	Completed	Transition when the Stage or Task instance completes normally. For a Stage instance, this means that all its child Task and Stage instances have reached a terminal or semi-terminal state (all child Task and Stage instances have reached disabled, terminated, completed, or fault). For a Task instance, this means its purpose has been accomplished (CaseTask instances have launched a new Case instance; ProcessTask instances have launched a Process instance and if output parameters are required then the Case or Process instance has completed and returned the output parameters; HumanTask instances have been completed by a human; etc.)
terminate	Active	Terminated	Transition by Case worker (human) decision or propagation from outer Stage instance. For a Stage instance, this state MUST propagate to all its contained EventListener, Milestone, Stage and Task instances.
exit	Available Active Enabled Disabled, Suspended	Terminated	Transition when the exit criteria of the Stage or Task instance becomes "true", or when the parent Stage instance transitions to Terminate state. This transition may represent a normal or an abnormal termination.

Transition	From	To	Description
	Failed		
resume	Suspended	Active	Transition by `Case` worker (human) decision or propagation from outer Stage instance. For a `Stage` instance, this state MUST propagate to all its contained `EventListener`, `Milestone`, `Stage` and `Task` instances.
re-activate	Failed	Active	Transition by the systems, an administrator, or by `Case` worker (human) when the source of the failure has been resolved
re-enable	Disabled	Enabled	Transition by a `Case` worker (human) decision
parent suspend	Available Active Enable Disabled	Suspended	Transition to Suspended when the parent `Stage` instance transitions to Suspended. `Stage` instances MUST propagate this state down to all its children.
parent resume	Suspended	Available Active Enable Disabled	Transition to the state previous to be suspended, when the parent stage transition out of Suspened. Stages propagate this state down to all its children

`Stage` instances propagate down some of their states, as follows:

Table 52: Stage instance state top-down propagation

When Stage moves into state		Child Stages and Tasks transition as follows			Child Milestones or Event Listeners transition as follows		
Transition	**Enter state**	**Transition**	**From state**	**To state**	**Transition**	**From state**	**To state**
create, parent resume	Available	--	Ø	Ø	--	Ø	Ø
enable, re-enable, parent resume	Enabled	--	Ø	Ø	--	Ø	Ø
disable, parent resume	Disabled	--	Ø	Ø	--	Ø	Ø
start, manual start	Active	create	Ø	Available	create	Ø	Available
resume, parent resume	Active	parent resume	Available	Suspended	N/A	Available	<impossible>
resume, parent resume	Active	parent resume	Enabled	Suspended			
resume, parent resume	Active	parent resume	Disabled	Suspended			

resume, parent resume	Active	parent resume	Active	Suspended			
resume, parent resume	Active	parent resume	Suspended	Suspended	resume	Suspended	Available
resume, parent resume	Active	--	Failed	Failed(1)			
resume, parent resume	Active	--	Completed	Completed	--	Completed	Completed
resume, parent resume	Active	--	Terminated	Terminated	--	Terminated	Terminated
suspend, parent suspend	Suspended	parent suspend	Available	Suspended	suspend	Available	Suspended
suspend, parent suspend	Suspended	parent suspend	Enabled	Suspended			
suspend, parent suspend	Suspended	parent suspend	Disabled	Suspended			
suspend, parent suspend	Suspended	parent suspend	Active	Suspended			
suspend, parent suspend	Suspended	--	Suspended	Suspended	--	Suspended	Suspended
suspend, parent suspend	Suspended	--	Failed	Failed(1)			
suspend, parent suspend	Suspended	--	Completed	Completed	--	Completed	Completed
suspend, parent suspend	Suspended	--	Terminated	Terminated	--	Terminated	Terminated
fault	Failed	--	Available	Available	--	Available	Available
fault	Failed	--	Enabled	Enabled			
fault	Failed	--	Disabled	Disabled			
fault	Failed	--	Active	Active			
fault	Failed	--	Suspended	Suspended	--	Suspended	Suspended
fault	Failed	--	Failed	Failed			
fault	Failed	--	Completed	Completed	--	Completed	Completed
fault	Failed	--	Terminated	Terminated	--	Terminated	Terminated
complete	Completed	N/A	Available	<impossible>	N/A	Available	Available
complete	Completed	N/A	Enabled	<impossible>			
complete	Completed	--	Disabled	Disabled			
complete	Completed	N/A	Active	<impossible>			
complete	Completed	N/A	Suspended	<impossible>	N/A	Suspended	Suspended
complete	Completed	--	Failed	Failed			
complete	Completed	--	Completed	Completed	--	Completed	Completed
complete	Completed	--	Terminated	Terminated	--	Terminated	Terminated

exit, terminate	Terminated	exit	Available	Terminated	parent terminate	Available	Terminated
exit, terminate	Terminated	exit	Enabled	Terminated			
exit, terminate	Terminated	exit	Disabled	Terminated			
exit, terminate	Terminated	exit	Active	Terminated			
exit, terminate	Terminated	exit	Suspended	Terminated	parent terminate	Suspended	Terminated
exit, terminate	Terminated	exit	Failed	Terminated			
exit, terminate	Terminated	--	Completed	Completed	--	Completed	Completed
exit, terminate	Terminated	--	Terminated	Terminated	--	Terminated	Terminated

Notes

(1) If the exception is fixed and the restart transition is taken to Active, then it should continue transition into Suspended state.

7.3.3 EventListener and Milestone Lifecycle

The following diagram illustrates the lifecycle of an `EventListener` or `Milestone` instance:

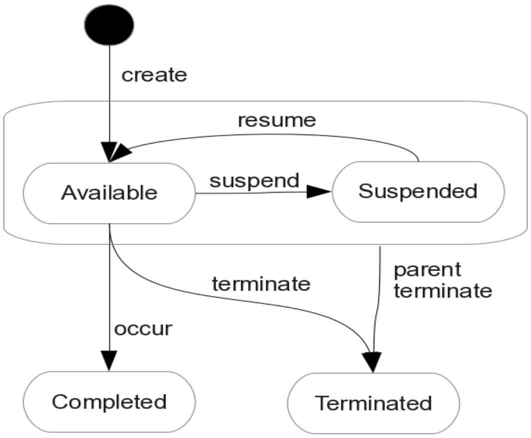


Figure 76: Lifecycle of an EventListener or Milestone instance

An `EventListener` or `Milestone` instance has the following states:

Table 53: EventListener and Milestone instance states

State	Description
Available	In this state an `EventListener` instance is waiting for the event to occur. A `Milestone` instance in this state is waiting for the `Sentry` (as entry criterion) to be satisfied.
Suspended	This state allows a `Case` worker (human) or an enclosing `Stage` instance to temporarily suspend an `EventListener` instance for which the event has not yet occurred, or to suspend a `Milestone` instance that has not been reached.

State	Description
Completed	Terminal state. For Events this state indicates that the `EventListener` instance was triggered, and that the event has been consumed. For `Milestone` instances this state indicates that one of the achieving criteria of the `Milestone` instance became “true”, i.e., that the Milestone has been achieved.
Terminated	Terminal state. This state indicates a termination by a `Case` worker (human) or an enclosing `Stage` instance, indicating that a `Case` worker (human) is not interest anymore on the event being listened to, or in the milestone being reached.

An `EventListener` or `Milestone` instance can undergo the following transitions:

Table 54: EventListener and Milestone instance transitions

Transition	From	To	Description
create	Ø	Available	Transition to the initial state (Available) when an `EventListener` or `Milestone` instance is created. For a `Milestone` instance, the `RepetitionRule` and `RequiredRule` Boolean expression MUST be evaluated in this transition, and their Boolean value SHOULD be maintained for the rest of the life of the `Milestone` instance.
suspend	Available	Suspended	Transition by `Case` worker (human) decision or propagation from outer `Stage` instance.
terminate	Available	Terminated	Transition by `Case` worker (human) decision or propagation from outer `Stage` instance.
occur	Available	Completed	For event listener transitions when the event being listened by the `EventListener` instance does occurs. For a `UserEventListener` instance this transition happens when a `Case` worker (human) decides to raise the event. For `Milestone` instance transitions when one of the achieving `Sentries` (entry critera) is satisfied.
resume	Suspended	Available	Transition by `Case` worker (human) decision or propagation from outer `Stage` instance.
parent terminate	Available Suspend	Terminated	Transition when the parent stage transition to terminate.

7.4 Sentry

When multiple entry criteria (sentries) are used only one is required to trigger the transition of the `Stage` or `Task` instance out of Available state. The same is true for exit criteria. When multiple exit criteria (sentries) are used only one is required to trigger to transition the `Stage` or `Task` instance from Active to Terminated.

A `Sentry's onPart` is satisfied when one of the following conditions is satisfied:

- For a `PlanItemOnPart`, its `Sentry` referred by `sentryRef` has occurred.
- For a `PlanItemOnPart` or `CaseFileItemOnPart`, its `sourceRef` transitions into the transition

described by the standardEvent (PlanItemTransition, or CaseFileItemTransition)

A Sentry is satisfied when one of the following conditions is satisfied:

- All of the onParts are satisfied AND the ifPart condition evaluates to “true”.
- All of the onParts are satisfied AND there is no ifPart.
- The ifPart condition evaluates to “true” AND there are no onParts.

7.5 Behavior Property Rules

Dynamically evaluated rules are used to derive Boolean values that can influence the execution of a Case instance. These are called, collectively, *Behavior Property Rules*. These rules are:

- Applicability rule (see 5.4.9.3)
- Stage.autocomplete (see 5.4.8)
- AutomaticActivationRule (see 5.4.11.1)
- RequiredRule (see 5.4.11.2)
- RepetitionRule (see 5.4.11.3)

In this section we consider how the semantics of these rules is related to transitions of the lifecycles of EventListener, Milestone, Stage resp. Task instances.

7.5.1 Stage.autoComplete

The following table describes the termination criteria of Stage instances based on the autoComplete attribute.

Table 55: Stage instance termination criteria

	autoComplete = true	autoComplete = false
Stage instance completion criteria	There are no Active children, AND all required (requiredRule evaluates to “true”) children are in {Disabled, Completed, Terminated, Failed}	There are no Active children AND (all children are in {Disabled, Completed, Terminated, Failed} AND there are no DiscretionaryItems) OR (Manual Completion AND all required (requiredRule evaluates to “true”) children are in {Disabled, Completed, Terminated, Failed})

In other words, a Stage instance SHOULD complete if a user has no option to do further planning or work with the Stage instance.

7.5.2 AutomaticActivationRule

The AutomaticActivationRule determines whether the Task or Stage instance should move to state Enabled or Active. This rule is evaluated and used when one of the entry criterion of the Task or Stage instance is satisfied. If this rules evaluate to “true” the Task or Stage instance transitions from Available to Active, otherwise it transitions from Available to Enabled. This rule impacts Stage or Task instances in Available state.

7.5.3 RequiredRule

The RequiredRule determines whether the Milestone, Stage or Task instance having this condition MUST be in the Completed, Terminated, Failed or Disabled state in order for its parent Stage instance to transition into the Completed state. This rule MUST be evaluated when the Milestone, Stage or Task instance is instantiated and transitions to the Available state, and their Boolean value SHOULD be maintained for the rest of the life of the Milestone, Stage or Task instance. If this rule is not present, then it is considered “false”. If this rule evaluates to “true”, the parent Stage instance MUST NOT transition to Complete state unless this Milestone, Stage or Task instance is in the Completed, Terminated, Failed or Disabled state. This rule impacts Stage instances in Available state.

7.5.4 RepetitionRule

This rule MUST be evaluated when the Milestone, Stage or Task instance is instantiated and transitions to the Available state, and their Boolean value SHOULD be maintained for the rest of the life of the Milestone, Stage or Task instance.

Stage and Task instances with a RepetitionRule evaluating to “true” will create an instance every time an entry criterion with an onPart is satisfied. Under that condition a new instance is created and because the entry criteria is satisfied it moves from the Available state to either Active or Enabled state depending on the AutomaticActivationRule.

7.5.5 ApplicabilityRule

This rule is evaluated and used for planning. It impacts planning by a HumanTask or into a Stage instance. During planning the only DiscretionaryItems that MUST be shown to the Case Worker (in the authorizedRoleRef) are those, for which the ApplicabilityRule evaluates to “true”.

7.6 Planning

Planning is constrained to certain states in the lifecycle of the Stage or HumanTask instance as described in the following table.

Table 56: Planning constrained to Case, Stage and Task instance lifecycles

Contain a Planning Table	States for which planning is allow
casePlanModel	Active, Failed, Suspended, Completed, Terminated
Stage instance	Active, Available, Enabled, Disabled, Failed, Suspended
HumanTask instance	Active

If a Stage instance is in Active state, then the planned PlanItems are instantiated immediately after planning completes. If the Stage instance is in another valid planning state, the planned PlanItems are instantiated when the Stage instance transitions to Active state. When a Stage instance has a PlanningTable, the TableItems of that PlanningTable can be used for planning. The resulting instances of the planning MUST be added to the Stage instance.

Case workers planning at a particular HumanTask instance are constrained to use the PlanningTable for that HumanTask instance. The resulting instances of the planning MUST be added to the parent Stage instance of the HumanTask instance. Those planned PlanFragments, Stages or Tasks are instantiated immediately after planning completes (because the parent Stage instance in which the planning task is taking place is in Active state).

7.7 Connector

Connectors are optional visual elements only and do not have associated execution semantics.

8 Exchange Formats

8.1 Interchanging Incomplete Models

It is common for `Case` models to be interchanged before they are complete. This occurs frequently when doing iterative modeling, where one user (such as a subject matter expert or business user) first defines a high-level model and then passes it on to another person to complete or refine the model.

Such "incomplete" models are ones in which not all of the mandatory model attributes have been filled in yet or the cardinality lower bound of attributes and associations has not been satisfied.

XMI allows for the interchange of such incomplete models. In CMMN, we extend this capability to interchange of XML files based on the CMMN XML-Schema. In such XML files, implementers are expected to support this interchange by:

- Disregarding missing attributes that are marked as "required" in the CMMN XML-Schema.
- Reducing the lower bound of elements with "minOccurs" greater than 0.

8.2 Machine Readable Files

CMMN 1.0 machine-readable files, including XSD and XMI files can be found in OMG Document bmi/2012-11-05, which is a zip file containing all the files:

- XML-Schema (XSD) files are found under the XSD folder of the zip file, the main file is XSD/CMMN10.xsd
- XMI files are found under the XMI folder of the zip file, the main file is XMI/CMMN10.xmi

8.3 XSD

8.3.1 Document Structure

A domain-specific set of `Case` model elements is interchanged in one or more CMMN files. The root element of each file MUST be <cmmn:definitions>. The set of files MUST be self-contained, i.e. all definitions that are used in a file MUST be imported directly or indirectly using the <cmmn:import> element.

Each file MUST declare a "targetNamespace" that MAY differ between multiple files of one `Case` model. CMMN files MAY import non-CMMN files (such as XSD's and BPMN files) if the contained elements use external definitions.

8.3.2 References within CMMN XSD

All CMMN elements contain ID's and within the CMMN XML-Schema, references to elements are expressed via these ID's. The XML-Schema IDREF (for a reference with multiplicity 1) and IDREFS (for references with multiplicity greater than 1) types are the traditional mechanisms used for referencing by ID's within a single XML file. The CMMN XSD supports referencing by ID, across files, by utilizing QNames. A QName consists of two parts: An (optional) namespace prefix and a local part. When used to reference a CMMN element, the local part is expected to be the ID of the element.

For example, consider the following Case

```
<case name=”Fraud Investigation” id=”Fraud_Investigation_Case_ID1”>
 …
</case>
```

When this Case is referenced from another file, the reference would take the following form

```
      caseRef=”case_ns:Fraud_Investigation_Case_ID1”
```

where “case_ns” is the namespace prefix associated with the case namespace upon import, and “Fraud_Investigation_Case_ID1” is the value of the ID attribute for the Case.

The CMMN XML-Schema utilizes IDREF and IDREFS wherever possible and resorts to QName only when references can span multiple files. In both situations however, the reference is still based on ID’s.