Date: December 2021

OBJECT MANAGEMENT GROUP®

Shared Data Model and Notation (SDMN)

V1.0 – beta 1

OMG Document Number: dtc/22-01-03

Normative:

Standard Document URL: https://www.omg.org/spec/SDMN

Machine Consumable file(s):

---

https://www.omg.org/spec/SDMN/20211108/SDMN.xmi
https://www.omg.org/spec/SDMN/20211108/SDMNDI.xmi
https://www.omg.org/spec/SDMN/20211108/SDMN.xsd
https://www.omg.org/spec/SDMN/20211108/SDMN-Semantic.xsd
https://www.omg.org/spec/SDMN/20211108/SDMNDI.xsd
https://www.omg.org/spec/SDMN/20211108/SDMN-Libary.xml

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 https://www.omg.org, under Documents, Report a Bug/Issue.

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 https://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:

https://www.omg.org/spec

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
9C Medway Road
PMB 274
Milford, MA 01757
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 https://www.iso.org

Issues

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 https://www.omg.org, under Documents, Report a Bug/Issue.

1 Scope

A Shared Data Model is a repository of **DataItems** to be used (referenced) by the other BPM-Plus (**BPM+)** data elements:

- BPMN Data Objects, CMMN Case File Items, DMN Data Inputs, etc.
- The **DataItems** can be created once and maintained in a single location and then maintenance can be distributed across multiple models
    - This eliminates the manual synchronization burden of working with the **BPM+** models without a Shared Data Model
- A Shared Data Model is a model because there are relationships between the **DataItems** (e.g., parent-child)
    - A diagram is included to visualize the **DataItems** and their relationships
- Note that a Shared Data Model is not an executable model
    - It is a library for the executable **BPM+** models

The primary goal of **SDMN** is to provide a set of structural elements that are common to other Object Management Group (OMG) specifications. BPM+ Knowledge Package Model and Notation (**BKPMN)**, Pedigree and Provenance Model and Notation (**PPMN)**, and **SDMN** have been structured to be dependent on the elements defined in Specification Common Elements (**SCE** [OMG doc number bmi-2021-12-09]**)**. Other Business Modeling and Integration (BMI) Task Force and Healthcare Domain Task Force (HDTF) specifications may also utilize the elements of **SCE** as they are updated in the future.

2 Conformance

2.1 General

Software can claim compliance or conformance with **SDMN 1.0** if and only if the software fully matches the applicable compliance points as stated in the specification. In addition, the structural elements provided by Specification Common Elements (**SCE 1.0** [OMG doc number bmi-2021-12-09]) are also required in a compliant or conformant software solution. Software developed only partially matching the applicable compliance points can claim only that the software was based on this specification but cannot claim compliance or conformance with this specification.

2.2 Shared Data Modeling Conformance

The implementation claiming conformance to the Shared Data Modeling Conformance SHALL comply with all of the requirements set forth in Clauses 8**Error! Reference source not found.**, 9, and 10; and it should be conformant with the Visual Notation Conformance in Clause 14. Conformant implementations SHALL fully support and interpret the exchange format specified in Clause 13.

This compliance point is intended to be used by **SDMN** modeling tools.

2.3 Visual Conformance

An implementation that creates and displays **SDMN** models SHALL conform to the specifications and restrictions with respect to diagrammatic relationships between graphical elements, as described in Clause **Error! Reference source not found.**. A key element of **SDMN** is the choice of shapes and icons used for the graphical elements identified in this specification. The intent is to create a standard visual language that all Shared Data modelers will recognize and understand. An implementation that creates and displays **SDMN** models SHALL use the graphical elements, shapes, markers and decorators illustrated in this specification.

There is flexibility in the size, color, line style, and text positions of the defined graphical elements, except where otherwise specified. In particular:

- **SDMN** elements MAY have labels (e.g., its name and/or other attributes) placed inside the shape, or above or below the shape, in any direction or location, depending on the preference of the modeler or modeling tool vendor.
- The fills that are used for the graphical elements MAY be white or clear. The notation MAY be extended to use other fill colors to suit the purpose of the modeler or tool (e.g., to highlight the value of an object attribute).
- Graphical elements, shapes, and decorators MAY be of any size that suits the purposes of the modeler or modeling tool with the condition that the additional graphical elements SHALL NOT conflict with any current BPM+ Standard defined graphical element.
- The lines that are used to draw the graphical elements MAY be black.
  - The notation MAY be extended to use other line colors to suit the purpose of the modeler or tool (e.g., to highlight the value of an object attribute).
  - The notation MAY be extended to use other line styles to suit the purpose of the modeler or tool (e.g., to highlight the value of an object attribute) with the condition that the line style SHALL NOT conflict with any current BPM+ Standard defined line style.

The following extensions to a **SDMN** model are permitted:

- New decorators or indicators MAY be added to the specified graphical elements. These decorators or indicators could be used to highlight a specific attribute of a **SDMN** element or to represent a new subtype of the corresponding concept with the condition that the additional graphical elements SHALL NOT conflict with any current BPM+ Standard defined decorator or indicator.
- A new shape representing a kind of **DataItem** MAY be added to a model with the condition that the shape SHALL NOT conflict with the shape specified for any other BPM+ Standard element or decorator.
- Graphical elements MAY be colored, and the coloring MAY have specified semantics that extend the information conveyed by the element as specified in this standard.
- The line style of a graphical element MAY be changed, but that change SHALL NOT conflict with any other line style REQUIRED by this specification or the other BPM+ Standards.
- An extension SHALL NOT change the specified shape of a defined graphical element or decorator. (e.g., changing a square into a triangle, or changing rounded corners into squared corners, etc.).

This compliance point is intended to be used by entry-level **SDMN** tools.

3 References

3.1 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

- Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, IETF RFC 2119, March 1997 http://www.ietf.org/rfc/rfc2119.txt
- [BPMN] OMG Business Process and Model Notation (BPMN™): https://www.omg.org/bpmn/
- [CMMN] OMG Case Management Model and Model Notation (CMMN™): https://www.omg.org/spec/CMMN/
- [DD] Diagram Definition (DD™)
- [DMN] OMG Decision Model and Model Notation (DMN™): https://www.omg.org/spec/DMN/
- [MOF] Meta Object Facility (MOF™): https://www.omg.org/spec/MOF/

- [SCE] Specification Core Elements (SCE): https://www.omg.org/spec/SDMN/
- [UML] Unified Modeling Language ™ (UML®): http://www.omg.org/spec/UML
- [XMI] XML Metadata Interchange (XMI®) http://www.omg.org/spec/XMI

3.2 Non-normative References

The following normative documents contain provisions which, through reference in this text, constitute exemplars or influencers of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

- [MDMI] OMG Model Driven Message Interoperability (MDMI), Version 1.0: https://www.omg.org/spec/MDMI/
- [SysML] OMG Systems Modeling Language (SysML®): http://www.omg.org/spec/SysML/

4 Terms and Definitions

The table below presents a glossary for this specification:

Table 1. Glossary

Term	Definition
Case	A **CMMN** element that is a proceeding that involves actions taken regarding a subject in a particular situation to achieve a desired outcome.
DataItem	A **SDMN** DataItem represents a common definition and structure for the data handling elements of the other BPM+ models.
DataState	DataItemscan optionally reference a DataState element, which is the state of the data contained in the DataItem. The definition of these DataStates, e.g., possible values and any specific semantic are out of scope of this specification. Therefore, SDMN adopters can use the DataState element and the SDMN extensibility capabilities to define their DataStates.
Decision	A **DMN** element that is the act of determining an output value (the chosen option), from a number of input values, using logic defining how the output is determined from the inputs.
ItemDefinition	Defines the detailed structure, which can be simple or complex, of a DataItem.
Process	A **BPMN** element that describes a sequence or flow of Activities in an organization with the objective of carrying out work. The ProcessRef element provides a link to a Process in a **BPMN** document.

5 Symbols

There are no symbols defined in this specification.

6 Additional Information

6.1 Conventions

The section introduces the conventions used in this document. This includes (text) notational conventions and notations for schema components. Also included are designated namespace definitions.

6.2 Typographical and Linguistic Conventions and Style

This document incorporates the following conventions:

- The keywords "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in this document are to be interpreted as described in RFC-2119.
- A **term** is a word or phrase that has a special meaning. When a term is defined, the term name is highlighted in **bold** typeface.
- A reference to another definition, section, or specification is highlighted with underlined typeface and provides a link to the relevant location in this specification.
- A reference to a graphical element is highlighted with a bold, capitalized word (e.g., **ProcessRef**).
- A reference to a non-graphical element or **SDMN** concept is highlighted by being italicized and (e.g., *Documentation*).
- A reference to an attribute or model association will be presented with the `Courier New` font (e.g., `Expression`).
- Non-normative examples are set off in boxes and accompanied by a brief explanation.
- XML and pseudo code is highlighted with `Courier New` typeface. Different font colors MAY be used to highlight the different components of the XML code.
- The cardinality of any content part is specified using the following operators:
  - [1] — exactly once
  - [0..1] — 0 or 1
  - [0..*] — 0 or more
  - [1..*] — 1 or more
- Attributes separated by | and grouped within { and } — alternative values
  - <value> — default value
  - <type> — the type of the attribute

6.3 Display of Metamodel Diagrams

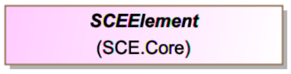
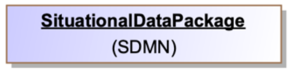
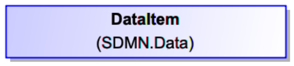
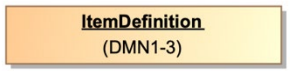
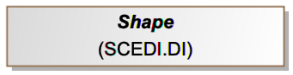
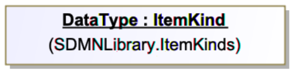
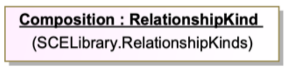
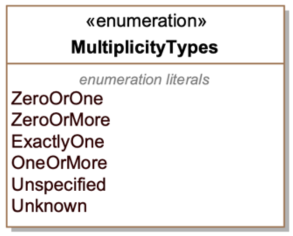
The metamodel presented in these sections utilizes the patterns and mechanisms that are used for the current **BPM+** specifications. **BPM+** specifications rarely display the entire metamodel of a technical specification in a single diagram. The entire metamodel would be very large, complicated, and hard to follow. Typically, a specification will present sub-sets of the overall metamodel as they apply to specific topics. For example, in the **BPMN** specification there are metamodel diagrams that show the elements relating to activities or data elements. This document will follow that pattern and present sub-sets of a larger metamodel.

The metamodel diagrams are Unified Modeling Language (UML) structure diagrams. In addition to the metamodel, OMG specifications provide XML schemas which map to the metamodels. In general, it is through XML documents that **BPM+** models are stored and exchanged.

Further, some of the metamodel elements are references to elements from other specifications. To clarify the owner of the metamodel element, there is a parenthesized text that identifies the model owner of that element. In addition, colors are used to support the text identification of the owner-language of that element. The colors are used as an aid to distinguish the languages but does not represent a normative aspect of the metamodels nor do they add any semantic information about the metamodels.

The table below presents examples of elements used throughout the metamodel diagrams within this specification:

Table 2. SDMN Metamodel Color-Coding

Element	Description	Example Color
SCE Structural Class	Metamodel elements from the **SCE 1.0** specification [OMG doc number bmi-2021-12-09] are shown in **SDMN** metamodel diagrams when **SDMN** elements are dependent on a **SCE** element. These elements include the owner of the language (SCE) in parenthases below the element name and these elements are color-coded lavender (see figure to the right).	
SDMN General Class	These elements include the owner of the language (SDMN) in parenthases below the element name and these elements are color-coded purple and the border line color is purple (see figure to the right). These make up the majority of metamodel elements shown in this specification.	
SDMN General Class (focus of diagram)	These elements have the same naming and color, but the border line color is dark blue instead of light brown (see figure to the right). They are highlighted as the focus of the particular metamodel diagram. This is an informative depiction that does not add any semantic information about the particular metamodel diagram.	
DMN General Class	Metamodel elements from the **DMN 1.3** specification are shown in **SDMN** metamodel diagrams when **SDMN** elements are dependent on a **DMN** element. These elements include the owner of the language (DMN) in parenthases below the element name and these elements are color-coded yellowish (see figure to the right).	
External Class	Classes from specifications that are not specifically part of the BPM+ stack of standards can be included in metamodel diagrams and display the owner of the language in parenthases below the element name and these elements are color-coded light-gray. (see figure to the right).	
SDMN Class Instance	These elements elements include the owner of the language (SDMN) in parenthases below the element name and these elements are color-coded light-purple to identify **SDMN** class instances from the **SDMN** Library (see figure to the right).	
SCE Class Instance	These elements include the owner of the language (SCE) in parenthases below the element name and these elements are color-coded light-violet to identify **SCE** class instances from the **SCE** Library (see figure to the right).	 Composition : RelationshipKind (SCELibrary.RelationshipKinds)
Enumerations	(see figure to the right).	 «enumeration» MultiplicityTypes enumeration literals ZeroOrOne ZeroOrMore ExactlyOne OneOrMore Unspecified Unknown

6.4 Use of Text, Color, Size, and Lines in a Diagram

- Diagram elements MAY have labels (e.g., its name and/or other attributes) placed inside the shape, or above or below the shape, in any direction or location, depending on the preference of the modeler or modeling tool vendor.
- The fills that are used for the graphical elements MAY be white or clear.
  - The notation MAY be extended to use other fill colors to suit the purpose of the modeler or tool (e.g., to highlight the value of an object attribute).
- Diagram elements and markers MAY be of any size that suits the purposes of the modeler or modeling tool.
- The lines that are used to draw the graphical elements MAY be black.
  - The notation MAY be extended to use other line colors to suit the purpose of the modeler or tool (e.g., to highlight the value of an object attribute).
  - The notation MAY be extended to use other line styles to suit the purpose of the modeler or tool (e.g., to highlight the value of an object attribute) with the condition that the line style SHALL NOT conflict with any current defined line style of the diagram.

6.5 Abbreviations

The table below presents a list of acronyms, and their defintion, that are used in this specification:

Table 3. Acronyms

Acronym	Definition
BHMN	BPM+ Harmonization Model and Notation
BKPMN	BPM+ Knowledge Package Model and Notation
BPM+	Business Process Management Plus
BPMN	Business Process Model and Notation
CMMN	Case Management Model and Notation
DC	Diagram Commons
DD	Diagram Definition
DI	Diagram Interchange
DMN	Decision Model and Notation
MDMI	Model Driven Message Interoperability
MOF	Meta Object Facility
OMG	Object Management Group
PPMN	Provenance and Pedigree Model and Notation
RFC	Remote Function Call
SCE	Specification Common Elements
SDMNDI	Shared Data Model and Notation Diagram Interchange
SDMN	Shared Data Model and Notation
SysML	Systems Modeling Language
URI	Uniform Resource Identifier
XMI	XML Metadata Interchange
XML	Extensible Markup Language

6.6 Structure of this Document

This document provides a brief introduction to **SDMN** and its purpose (see the section entitled "**Error! Reference source not found.**"). The introduction is followed by normative clauses that define the elements of the specification and their properties and associations (see the sections entitled "SDMN Metamodel" (Clause 9); "SDMN Model

Elements” (Clause 10); “SDMN Models” (Clause 11); “SDMN Library” (Clause 12); “Mapping to BPM+ Models” (Clause 13); and “SDMN Diagram Interchange” (Clause 16)).

6.7 Acknowledgements

Submitting Organizations (RFP Process)

- Auxilium Technology Group, LLC
- BPM Advantage Consulting, Inc.

Supporting Organizations (RFP Process)

The following organizations support this specification but are not formal submitters:

- Airbus Group
- BookZurman, Inc.
- Camunda Services GmbH
- Department of Veterans Affairs
- FICO
- Mayo Clinic
- MDIX, Inc.
- Red Hat
- Thematix Partners, LLC
- Trisotech
- XZYOS, LLC

Special Acknowledgements

The following persons were members of the core teams that contributed to the content of this specification: John Butler, Keith Butler, Lloyd Duggen, Denis Gagne, Eder Ignatowicz, Peter Haug, Elisa Kendall, Matteo Mortari, Falko Menge, Sean Muir, Robert Lario, Ken Lord, Keith Salzman, Jane Shellum, Davide Sottara, and Stephen A. White.

7 Overview

The focus of this document is to define the content and structure of a Shared Data Model and Notation (**SDMN**).

A "Shared Data Model" (SDM) is a library of data elements that supports a set of BPM+ Models (see directly below) that are used together to address a particular business modeling topic. In particular, a Shared Data Model will provide a single source for the definition of all, and only, the data elements that are used across those correlated BPM+ models. Thus, the SDM provides a shared, scoped, and focused view that supports mutual interfaces between the models, as well as external data sources.

7.1 What Constitutes a BPM+ Model?

Three OMG standards – Business Process Model and Notation (**BPMN**); Case Management Model and Notation (**CMMN**); and Decision Model and Notation (**DMN**) – are often used together to model real-world business situations since they provide (for the most part) a good separation of concerns for Process, Case, and Decision. Thus, the three languages are often spoken about and written about in this context. The origins of the **BPM+**

acronym was to reduce the burden of referring to all three specifications in speech and in print. A single acronym to refer to the three languages is just simpler.

The idea of **BPM+** has since expanded to be a business modeling language stack that will gain new standards as they are developed. The standards that fit into that stack will be languages that address additional areas of concerns and can interact with, in one way or another, with at least one of the other **BPM+** languages. **SDMN** is a modeling standard designed to fit into to the **BPM+** stack. In this context, a Shared Data Model is considered the "fourth pillar" of a BPM-Plus (BPM+) Knowledge Package. The other three pillars being the BPM+ standards for Process, Case, and Decision. Additionally, new standards are being developed to fit in the BPM+ stack. These include BPM+ Knowledge Package Model and Notation (**BKPMN**) and Pedigree and Provenance Model and Notation (**PPMN**).

7.2 Why a Shared Data Model?

Based on experience with the current set of **BPM+** standards – **BPMN**, **CMMN**, and **DMN** – the need of a centralized library of **DataItems** was identified (see the use case described in Clause 14.1 as an illustration of the drivers of this need). For example, using BPM+ models to address a large topic, such as the behaviors of a healthcare clinical guideline (e.g., for hypertension or kidney disease) may result in dozens of individual Process, Case, and Decision models. Specific data elements are frequently used by multiple models across the three classes of **BPM+** model types (Process, Case, and Decision). To continue the hypertension example, a data element for "blood pressure" may be used within a Process, Case, and/or Decision. To ensure consistency and accuracy across the models of these large topics, the detailed structures (names and types) of the data elements should be synchronized across all the models that use them.

Since the development of the models of these large topics are lengthy and iterative, the detailed structures of the shared data elements are likely to change over time. Experience has shown that synchronizing the changes to data elements across multiple models, multiple times, is a burdensome maintenance requirement.

Thus, a need for a central data library for the data elements of a BPM+ Knowledge Package was identified. This library would serve as a central source for the development of data elements that would be referenced by the other **BPM+** models. This library should reflect the structure and capabilities of the current **BPM+** models data elements. The library should also include a diagram and modeling environment that is consistent with the data representations of the current **BPM+** modeling environments to ease the modeling experience as a modeler moves between the respective modeling tools.

In addition to these three new BPM+ standards, there is a sixth standard, Specification Core Elements (**SCE**), that provides a set of common modeling language elements, such as root element and basic packaging capabilities. Instead of defining these basic, non-language specific elements within each of the new languages, **BKPMN**, **PPMN**, and **SDMN** are built upon the structures provided by **SCE**.

The following figure illustrates the relationships between the old and new **BPM+** standards.

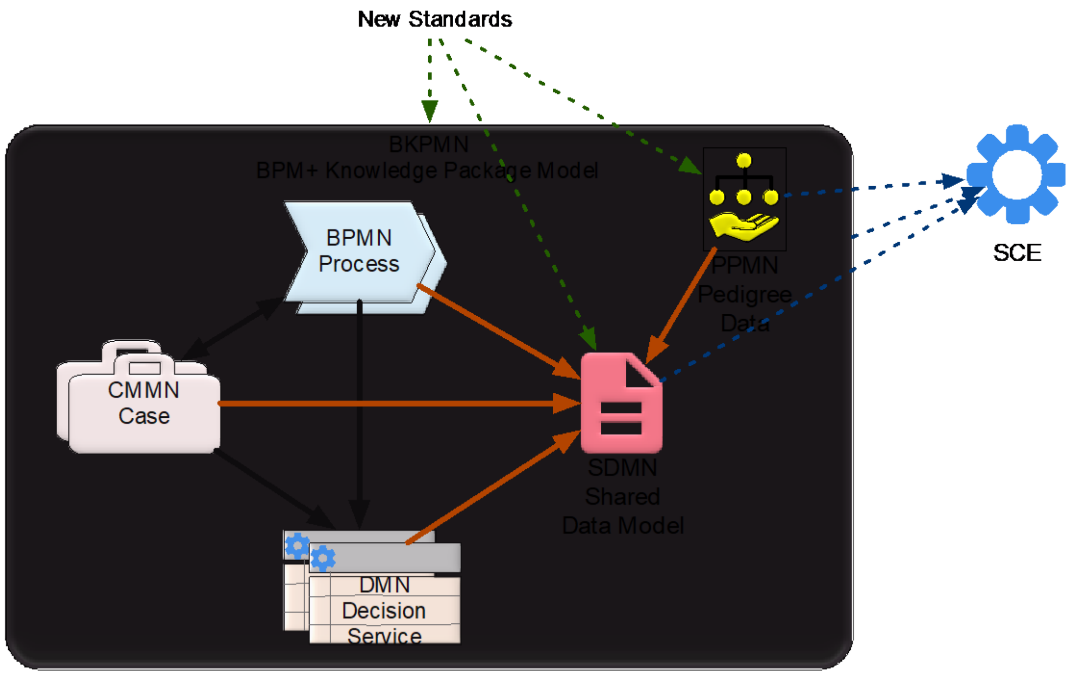


Figure 1: Overview of SDMN in the Context of BPM+ Standards

7.2.1 Use Case: Hello Patient

The BPM+ Health community has been defining Shareable Clinical Pathways by using the current **BPM+** standards to define formal and executable versions of current clinical guidelines (e.g., for hypertension, chronic kidney disease, etc.). Current clinical guidelines are usually found in printed or PDF documents and they contain vague and often confusing semantics leading to a great variability in how the guidelines are understood and performed.

This section describes a simple use case that was developed by the BPM+ Health community. At that time there was no concept of a BPM+ Knowledge Package or of a Shared Data Model. The work on this and other use cases was instrumental in identifying the need and requirements for a BPM+ Knowledge Package and a Shared Data Model.

Organizing BPM+ Models (A BPM+ Knowledge Package)

The use case defined the Processes, Cases, and Decision Services that are involved in managing a visit to a doctor's office. Note that these models were intended to be illustrative rather than an official, comprehensive healthcare guideline.

The following table lists the major **BPM+** model elements that made up the use case.

Table 4. List of BPM+ Models for the Hello Patient Use Case

Cases	Decision Services	Processes
1. Hello Patient 2. Perform Examination 3. Perform Additional Test for Physical	1. What is Treatment Plan? 2. What is Patient's BMI Category? 3. Weight Counseling Suggested? 4. What is Blood Pressure Rating? 5. Physical Required?	1. Manage Hello Patient Triggers 2. Evaluate Applicability 3. Manage Patient Visit 4. Check In Patient 5. Take Vital Signs 6. Check Out Patient 7. Update Appointment Information 8. Ask Screening Questions 9. Manage Counseling Referral

A larger use case for "Antenatal Care" was developed and contained more models than listed above. For that use case there were 9 Cases, 15 Decision Services, and 28 Processes.

Reviewing one of the models listed in the table above does not provide the overall scope and context of the set of models in the use case. While it may be possible to trace through the connections between the **BPM+** models, that tracing still does provide the proper context.

This lack of perspective resulted in a new type of diagram included with the use case. It is referred to as a Knowledge Diagram in this specification. The diagram provides graphical representations of the **BPM+** models and draws connectors to represent how the models can be traced through their connections. The following figure displays the Knowledge Diagram for the Hello Patient use case. Note that all the items listed in Table 1, above, have diagram elements associated with them. There are different notations for Processes, Cases, and Decision Services.

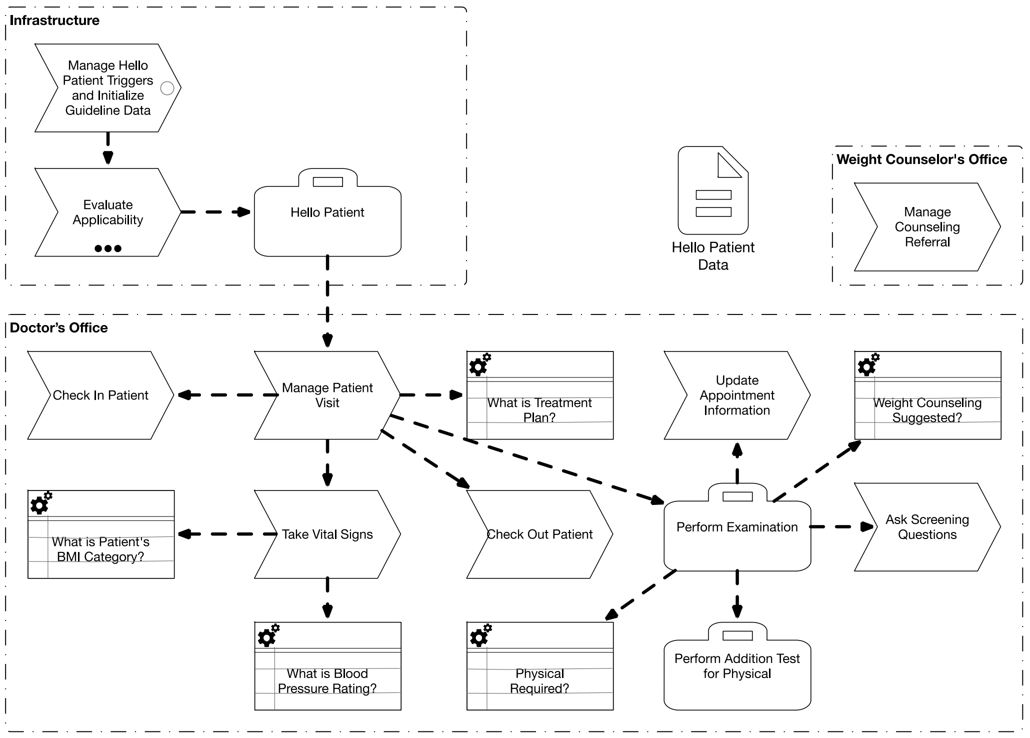

Infrastructure
Manage Hello Patient Triggers and Initialize Guideline Data
Evaluate Applicability
Hello Patient
Hello Patient Data
Weight Counselor's Office
Manage Counseling Referral
Doctor's Office
Check In Patient
Manage Patient Visit
What is Treatment Plan?
Update Appointment Information
Weight Counseling Suggested?
What is Patient's BMI Category?
Take Vital Signs
Check Out Patient
Perform Examination
Ask Screening Questions
What is Blood Pressure Rating?
Physical Required?
Perform Addition Test for Physical

Figure 2: Example of a BPM+ Knowledge Diagram

This diagram is just an example, and the exact notation is not a requirement for this specification, but there are requirements as to the type of elements, and how they are connected, listed below.

Note that the development of the Knowledge Diagram for the use cases was an indication that something else was needed to fully document the contexts of a set of **BPM+** models created for a specific topic. This and other factors led to the requirements for a BPM+ Knowledge Package.

Organizing BPM+ Data Elements (A Shared Data Model)

Several elements in BPM+ Models are intended to store or convey data required for the execution of those Models. **BPMN** has Data Objects, Data Inputs, Data Outputs, Data Stores, and Properties. **CMMN** has Case File Items. **DMN** has Information Items that are used for Data Inputs and Decisions. The Hello Patient use case employed many of these types of data elements within its **BPM+** models. The following table lists those data elements used within the set of **BPM+** models for the Hello Patient use case.

Table 5. List of Data Elements used by the BPM+ Models in the Hello Patient Use Case

Cases	Decision Services	Processes
1. Blood Pressure	**1. Blood Pressure**	**1. Blood Pressure**
2. Blood Pressure Goal	**2. Blood Pressure Goal**	**2. Blood Pressure Goal**
3. BMI Category	3. Blood Pressure Rating	3. Blood Pressure Rating
4. Encounter	**4. BMI Category**	**4. BMI Category**
5. Exam Data	5. Demographics	5. Demographics
6. Guideline Info	**6. Exam Data**	6. Encounter
7. Health Conditions	**7. Health Conditions**	**7. Exam Data**
8. Medication	**8. Medication**	**8. Health Conditions**
9. Medication Tolerances	**9. Medication Tolerances**	9. Loop Counter
10. Pathway Goals	10. Pathway Goals	**10. Medication**
11. Patient Health Record	11. Patient Complaints	**11. Medication Tolerances**
12. Referral	**12. Patient Health Record**	12. Pathway Goals
13. Treatment Plan	**13. Referral**	13. Patient Complaints
14. Vital Signs and Measurements	**14. Treatment Plan**	**14. Patient Health Record**
15. Weight Counseling Referral	15. Treatment Choice	**15. Referral**
16. Weight Counseling Referral Choice	**16. Vital Signs and Measurements**	**16. Treatment Plan**
	17. Weight Counseling Referral	17. Treatment Choice
	18. Weight Counseling Referral Choice	**18. Vital Signs and Measurements**
		19. Weight Counseling Referral

Note that the data elements listed in **bold** in the table are those that appear in all three types of **BPM**+ models. The other data elements appear in at least two of the model types.

The set of data elements listed in the above table reflect those data elements that are necessary for only the context of this use case (Hello Patient). They do not represent all the data elements that a doctor's office may require for all of its operations – let alone all the data elements required for the healthcare domain. The use case only specified the data elements that are shared across the models for its particular situation. Hence, we refer to sets of data elements used in this way as "Shared Data".

Since the use case employed all three different types of **BPM+** models (Process, Case, and Decision Service), the common data elements of the use case are shared and distributed across the three types of models. While there are some technical differences between how data is structured and used across the **BPM+** specifications, at the logical level, they all play the same role within the respective languages. This is evident when a specific conceptual data element (e.g., "Vital Signs and Measures") can be included in all three BPM+ modeling languages (see figure below). That is, the same data element (and its values during runtime) can be passed from a **CMMN** Case to a **BPMN** Process and then be used in a **DMN** Decision.

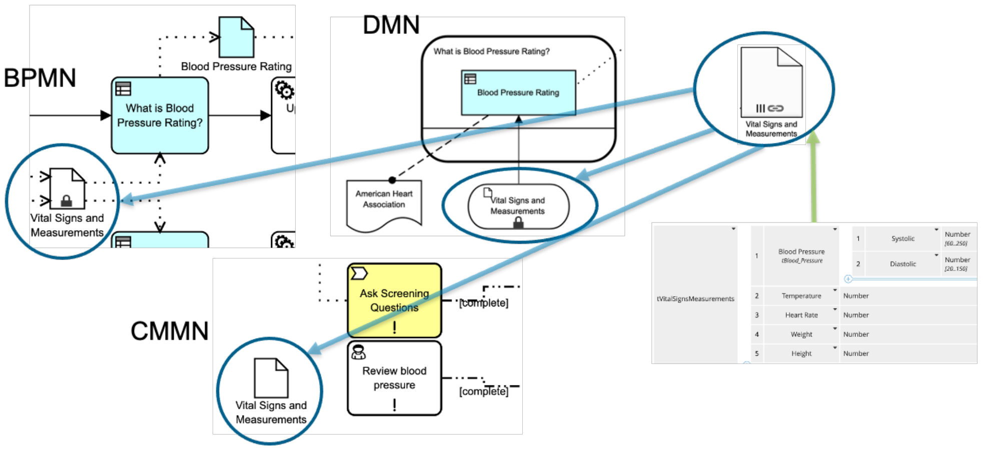

BPMN
Blood Pressure Rating
What is Blood Pressure Rating?
Vital Signs and Measurements
DMN
What is Blood Pressure Rating?
Blood Pressure Rating
American Heart Association
Vital Signs and Measurements
Vital Signs and Measurements
tVitalSignsMeasurements
1 Blood Pressure tBlood_Pressure
1 Systolic Number [60..250]
2 Diastolic Number [20..150]
2 Temperature Number
3 Heart Rate Number
4 Weight Number
5 Height Number
CMMN
Ask Screening Questions
[complete]
Review blood pressure
[complete]
Vital Signs and Measurements

Figure 3: Illustration of How Data Elements are Share Across BPM+ Models

Currently, the same data element has to be defined separately in the tools dedicated to each modeling language. There are no standard mechanisms for sharing data elements across the three types of **BPM+** models.

If there are a lot of data elements that are shared between the models of a BPM+ Knowledge Package, the development and maintenance burden for synchronizing the properties of the data elements will be problematic. All of the Hello Patient use date elements were used in at least two types of models. Each time any of the data elements were modified, which can happen multiple times during the BPM+ Knowledge Package development cycle, there would be one or more modifications in the other types of **BPM+** models. It would be up to the modeler to ensure that the modifications were made and were consistent.

This maintenance burden was the driver for defining a Shared Data Model, which would be a library of data elements that would readily be available for synchronization with the other **BPM+** models. That is, the **DataItems** of the Shared Data Model should share the same characteristics as the data elements of the three **BPM+** model data elements. Further, the modeling experience should be very similar across all four models to ease burdens on the modeler.

The Shared Data Model would provide an environment where data elements can be defined and modified in a single location and the changes could be distributed to the other **BPM+** models without additional work and vigilance by the modeler. Modeling tools that implement **SDMN** should provide a diagramming capability that is consistent with how current BPM+ modeling tools represent their data elements. Specifically, the notation for **BPMN** and **CMMN** data elements are consistent and should be used as the basis for a **SDMN** diagram. The following figure provides an example of how a **SDMN** Data Item Diagram could look.

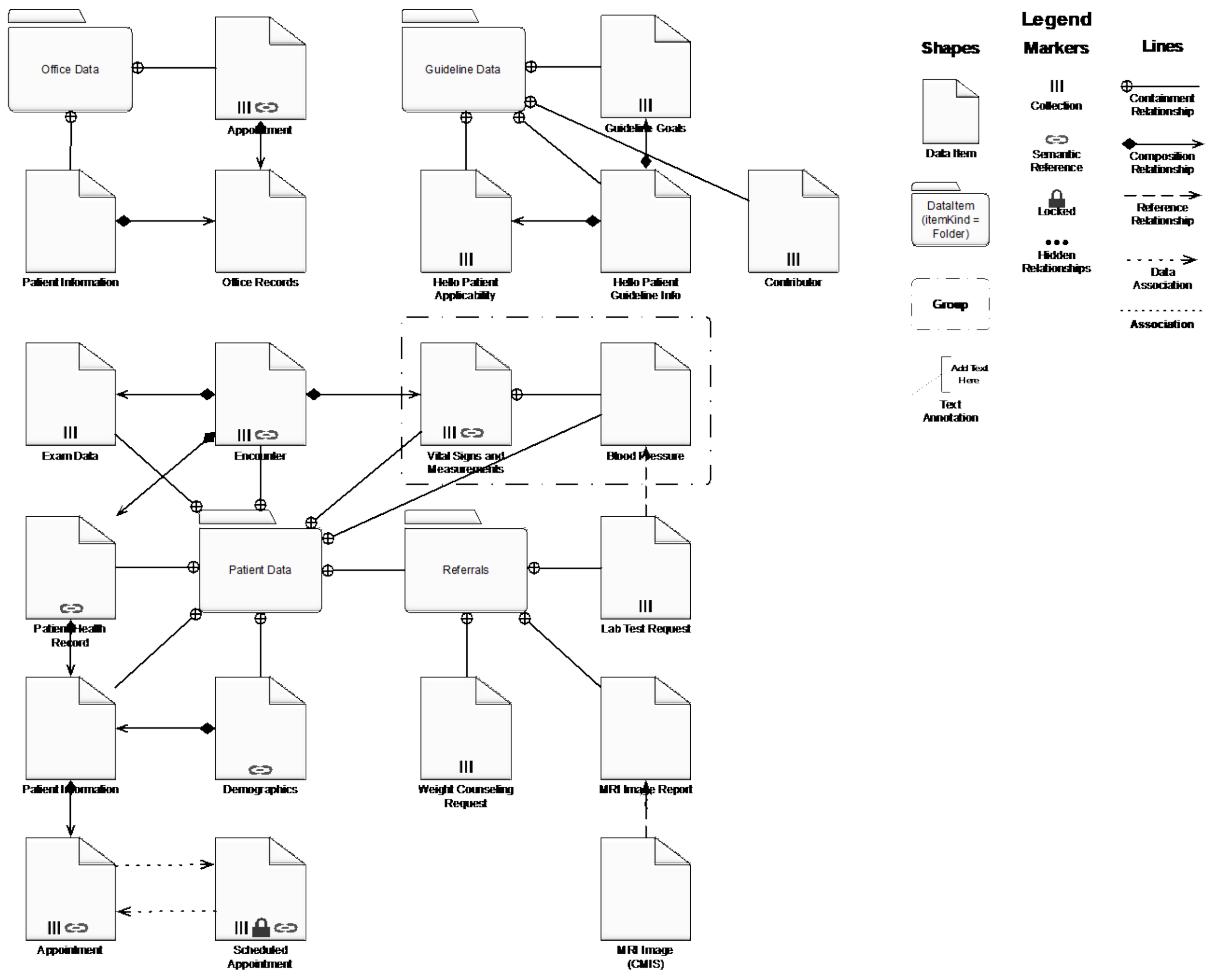


Figure 4: An Example of How a SDMN DataItem Diagram

A Shared Data Model would then become another component of a BPM+ Knowledge Package (as shown in Figure 2 above).

7.3 The Purpose and Use of a Shared Data Model in a BPM+ Knowledge Package

A **Shared Data Model** serves multiple purposes with a BPM+ Knowledge Package.

First, it provides a library of **DataItems** that serve as the source for the data elements of the **BPMN**, **CMMN**, and **DMN** models within the BPM+ Knowledge Package, including:

- **BPMN** Data Objects, Data Inputs, Data Outputs, and Messages
- **CMNN** Case File Items
- **DMN** Data Inputs and Decision Outputs

A Shared Data Model may also serve as a source for **BPMN** Data Object initialization at the start of a Process.

See the section "Pre-Assigning Values for DataItems," below, for more information.

8 Specification Core Elements

The **SDMN** specification utilizes (is dependent on) structural elements defined in the Specification Core Elements (**SCE**) metamodel. This metamodel is defined in a separate specification [OMG doc number bmi-2021-12-09] and contains a set of basic metamodel classes that are common to **BKPMN**, **PPMN**, and **SDMN** – and potentially other OMG specifications. Details about the elements of the **SCE** are maintained in a separate document.

As can be seen in the below, **SCE** defines elements that can be used by any modeling specification – that is, the elements are not specific to any particular area of concern, such a data, process, decision, etc. For example, the **SCE** *Documentation* element can be used (and is used) in any modeling specification since it is important to allow modelers to provide documentation about a semantic element they include in a model.

Because **SCE** defines these elements, **SDMN** does not have to duplicate them in this specification. **SDMN** can just create metamodel bindings to the elements in **SCE**. Thus, throughout this specification, **SCE** elements will be seen in metamodel diagrams and **SDMN** elements will be shown as being specializations of those **SCE** elements. The **SCE** and **SDMN** metamodel elements will be identified as described in Section 6.3.

The **SCE** high-level metamodel defines the basic infrastructure elements of a **BPM+** model (see figure below).

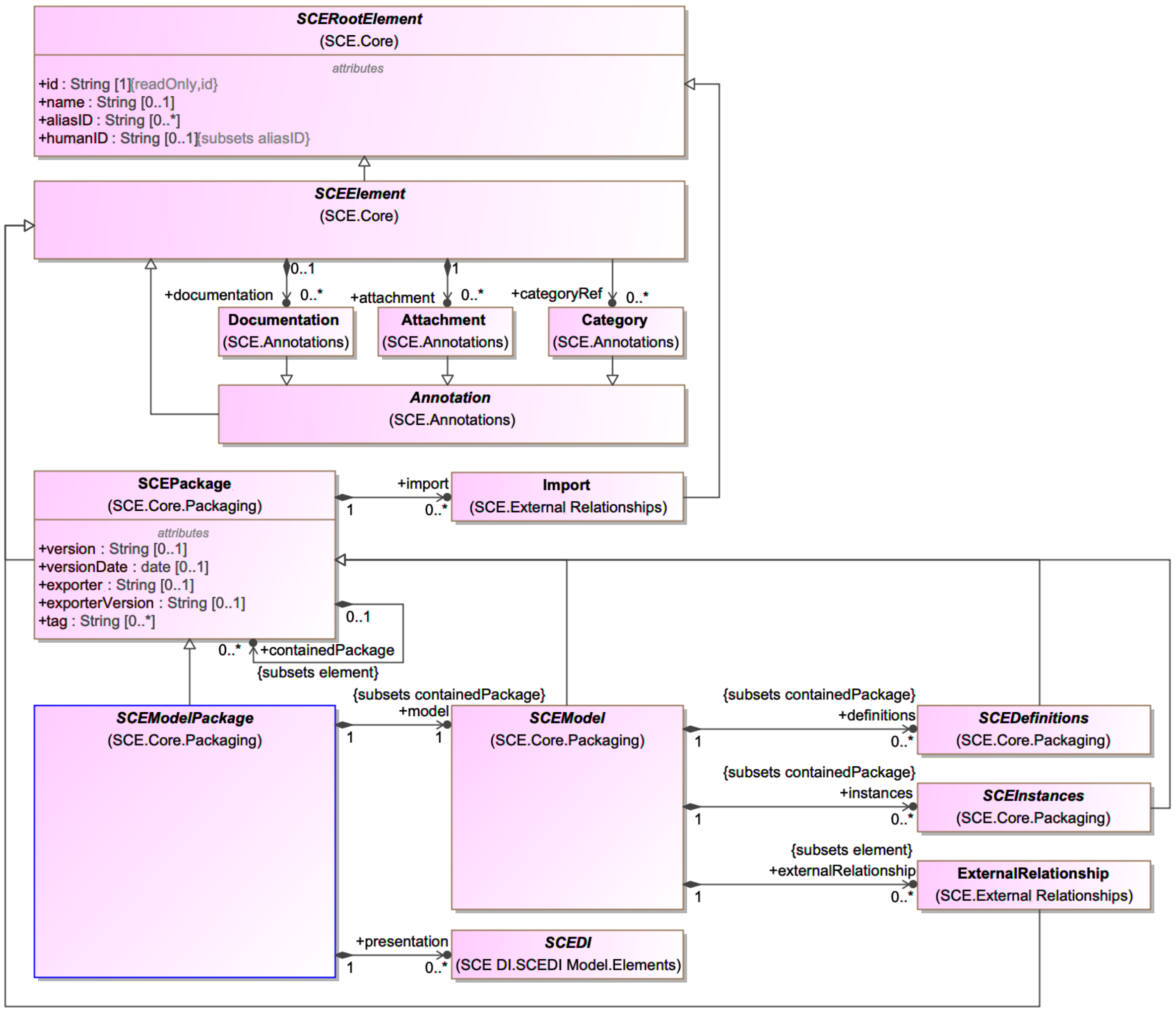


Figure 5: The Specification Core Elements (SCE) Base Metamodel

9 SDMN Metamodel

The **SDMN** core metamodel defines the basic infrastructure elements of a **Shared Data Model**. As mentioned in the previous section, **SDMN** is dependent on **SCE** [OMG doc number bmi-2021-12-09]. This dependency is manifested in multiple **SDMN** metamodel relationships. For example, the *SDMNModelPackage* element directly specializes the **SCE** *SCEModelPackage* element, thus inheriting all the properties and associations of that element.

The following figure shows the organization of the **SDMN** metamodel packages.

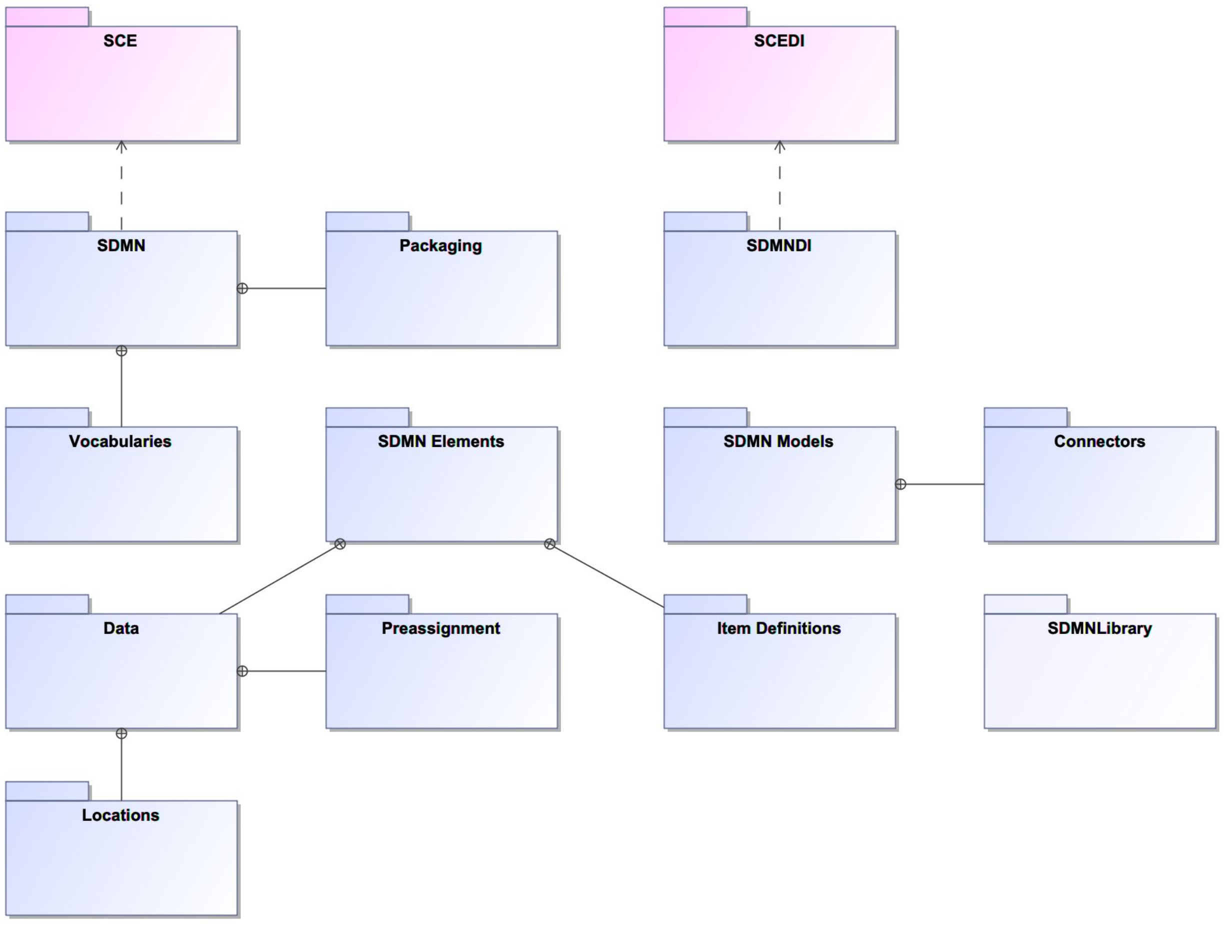


Figure 6: SDMN Main Packages

Further, most of the other **SDMN** elements directly specialize the **SCE** SCEElement. These relationships can be seen in the metamodel diagrams in this chapter as well as being identified in the "Generalizations" subsections for the relevant **SDMN** classes defined in this chapter.

The following figure shows the **SDMN** core elements metamodel.

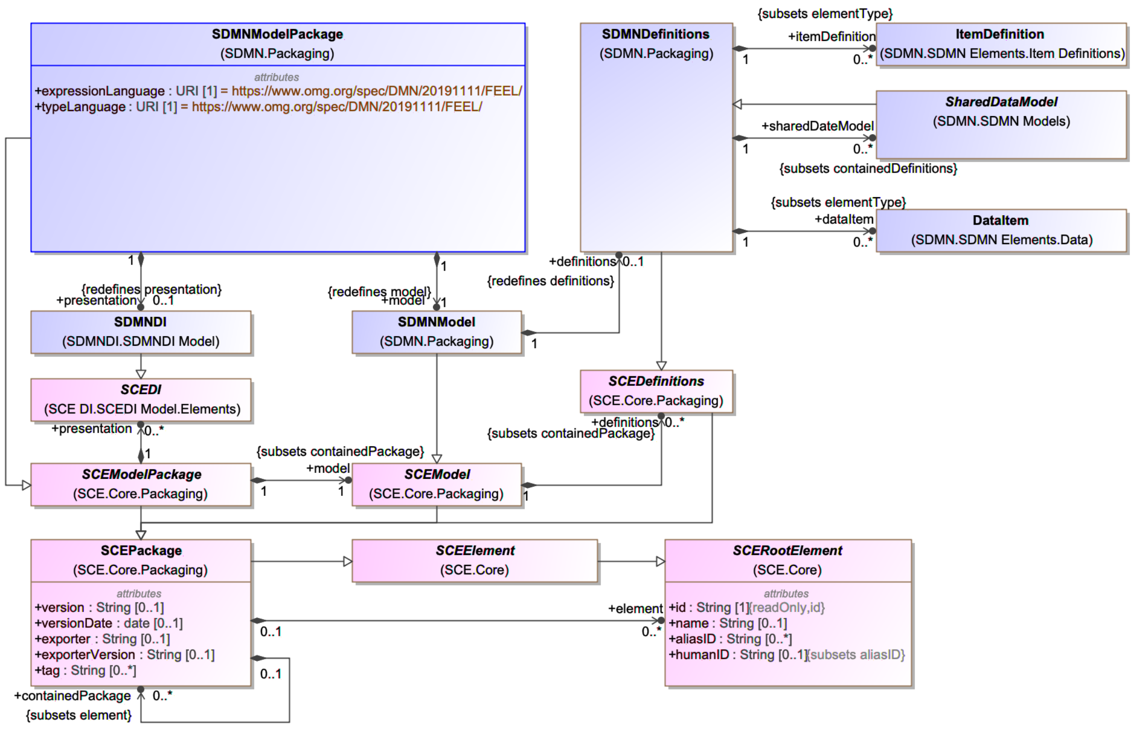


Figure 7: The SDMN Core Metamodel

9.1 Packaging

SDMN extends three of the five main packaging elements of **SCE** [OMG doc number bmi-2021-12-09]: *SCEModelPackage*, *SCEModel*, and *SCEDefinitions*. See the next three sections.

9.1.1 SDMNModelPackage

The *SDMNModelPackage* class is the outermost containing object for all **SDMN** elements. It defines the scope of visibility and the namespace for all contained elements. The interchange of **SDMN** files will always be through one or more *SDMNModelPackages*. Specifically, an XML file for a **Shared Data Model** usually would be appended with a ".sdmn" label.

The *SDMNModelPackage* contains two main elements: *SDMNModel* and *SDMNDI*.

The following figure shows the *SDMNModelPackage* metamodel.

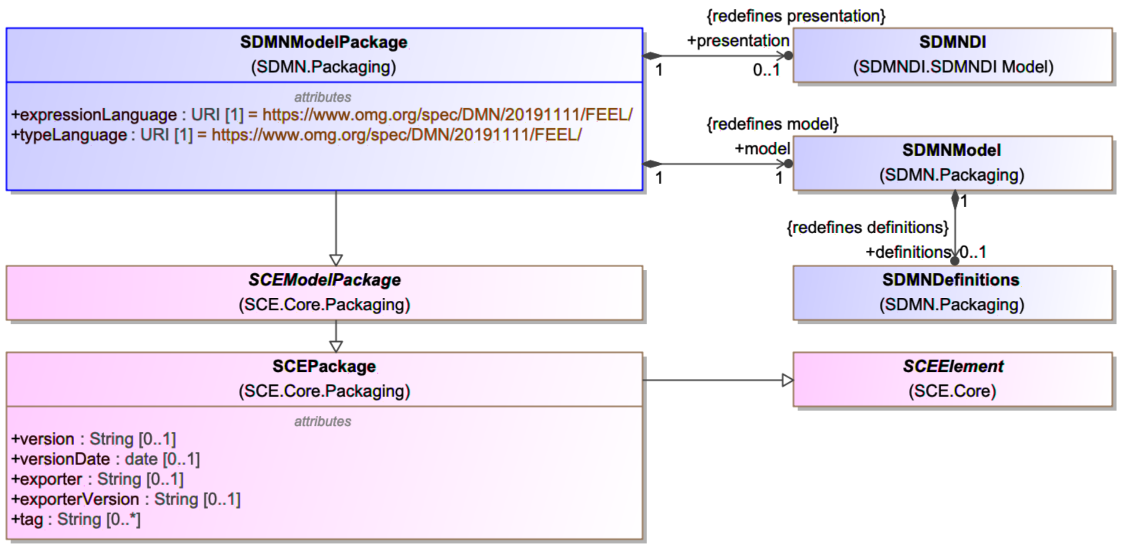


Figure 8: The SDMNModelPackage Metamodel

Generalizations

The *SDMNModelPackage* element inherits the attributes and/or associations of:

- *SCEModelPackage* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The following table presents the additional attributes and/or associations for *SDMNModelPackage*:

Table 6. SDMNModelPackage Attributes and/or Associations

Property/Association	Description
expressionLanguage : URI [1] default: https://www.omg.org/spec/DMN/20191111/FEEL/	This attribute identifies the formal expression language used in Expressions within the elements of this *SDMNModelPackage*. The Default is "http://www.omg.org/Spec/DMN/20180521/FEEL". This value MAY be overridden on each individual formal Expression. The language SHALL be specified in a URI format.
model : SDMNModel [1]	This the *SDMNModel* sub-package contained within a *SDMNModelPackage*. This is a subset of the `containedPackage` association of the *SCEPackage* element.
presentation : SDMNDI [0..1]	This attribute contains the Diagram Interchange information contained within this *SDMNModelPackage*. See the section entitled "SDMN Diagram Interchange" for more information.
typeLanguage : URI [1] default: https://www.omg.org/spec/DMN/20191111/FEEL/	This attribute identifies the type system used by the elements of this *SDMNModelPackage*. The Default is "http://www.omg.org/Spec/DMN/20180521/FEEL". This value can be overridden on each individual ItemDefinition. The language SHALL be specified in a URI format.

9.1.2 SDMNModel

The *SDMNModel* element contains the modeling content of an **SDMN** model (as opposed to the diagram interchange content for the modeling content). The two sub-packages for *SDMNModel* are *SDMNDefinitions* and *SDMNVocabulary*.

An *SDMNModel* is contained within an *SDMNModelPackage*.

The following figure shows the *SDMNModel* metamodel.

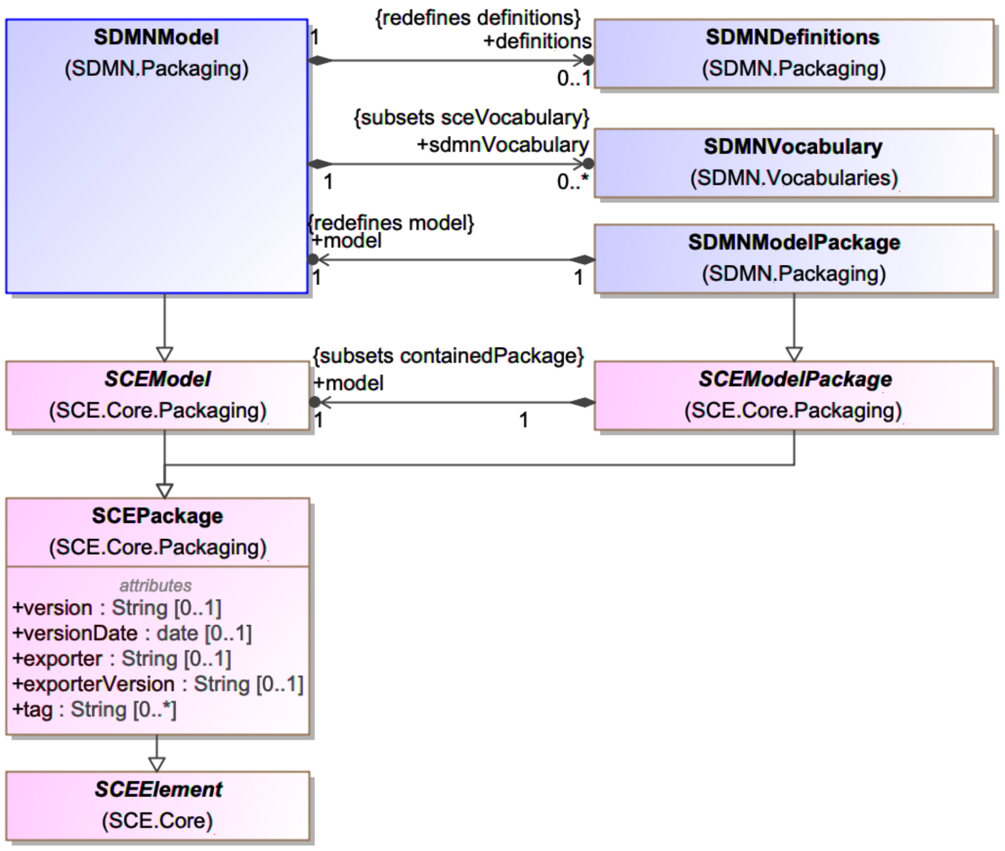


Figure 9: The SDMNModel Metamodel

Generalizations

The *SDMNModel* element inherits the attributes and/or associations of:

- *SCEModel* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The following table presents the additional attributes and/or associations for *SDMNModel*:

Table 7. SDMNModel Attributes and/or Associations

Property/Association	Description
definitions : SDMNDefinitions [0..*]	This is a list of the *SDMNDefinitions* that are included in the *SDMNModel*.

9.1.3 SDMNDefinitions

Most of the **SDMN** modeling elements are contained within the *SDMNDefinitions* sub-package. This includes the two key elements **DataItem** and *ItemDefinition*. It is contained within an *SDMNModel*.

The following figure shows the *SDMNDefinitions* metamodel.

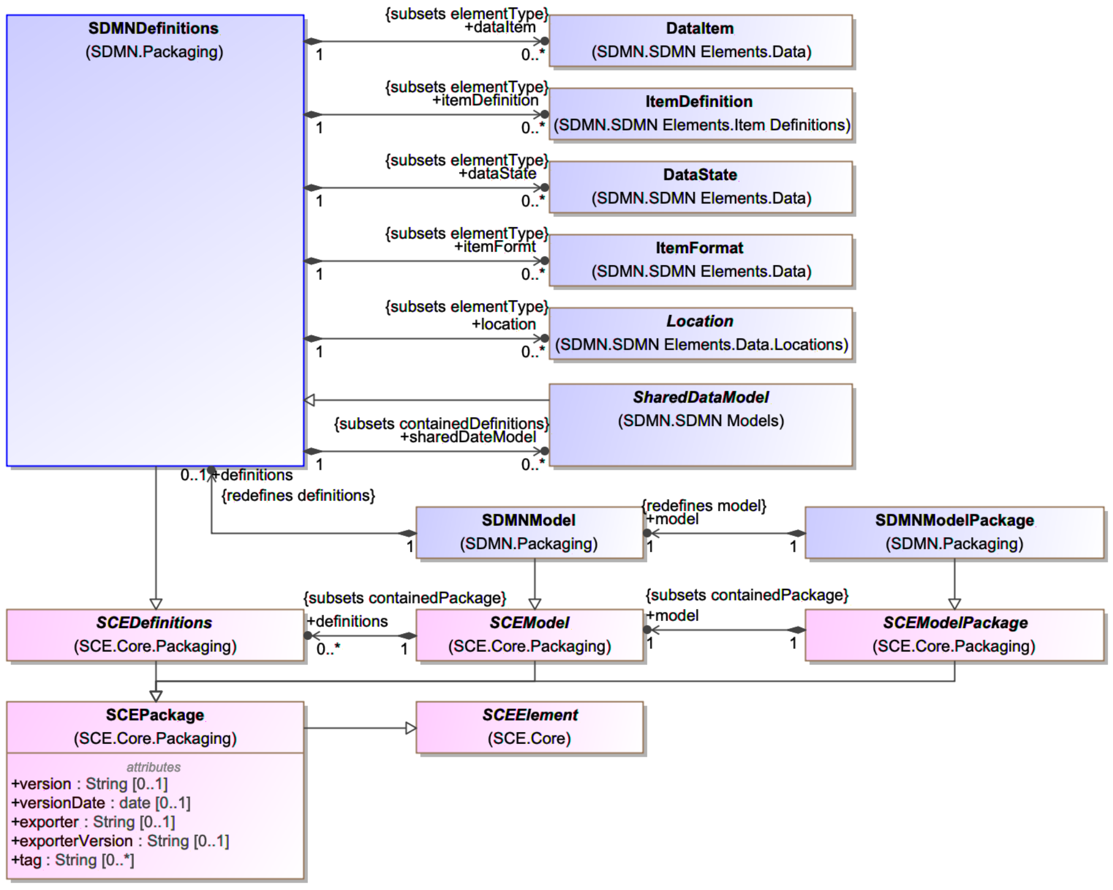

SDMNDefinitions (SDMN.Packaging)
{subsets elementType} +dataItem 1 0..*
DataItem (SDMN.SDMN Elements.Data)
{subsets elementType} +itemDefinition 1 0..*
ItemDefinition (SDMN.SDMN Elements.Item Definitions)
{subsets elementType} +dataState 1 0..*
DataState (SDMN.SDMN Elements.Data)
{subsets elementType} +itemFormt 1 0..*
ItemFormat (SDMN.SDMN Elements.Data)
{subsets elementType} +location 1 0..*
Location (SDMN.SDMN Elements.Data.Locations)
{subsets containedDefinitions} +sharedDateModel 1 0..*
SharedDataModel (SDMN.SDMN Models)
0..1 +definitions {redefines definitions} 1
SDMNModel (SDMN.Packaging)
{redefines model} +model 1 1
SDMNModelPackage (SDMN.Packaging)
SCEDefinitions (SCE.Core.Packaging)
{subsets containedPackage} +definitions 0..* 1
SCEModel (SCE.Core.Packaging)
{subsets containedPackage} +model 1 1
SCEModelPackage (SCE.Core.Packaging)
SCEPackage (SCE.Core.Packaging)
attributes
+version : String [0..1]
+versionDate : date [0..1]
+exporter : String [0..1]
+exporterVersion : String [0..1]
+tag : String [0..*]
SCEElement (SCE.Core)

Figure 10: The SDMNDefinitions Metamodel

Generalizations

The *SDMNDefinitions* element inherits the attributes and/or associations of:

- *SCEDefinitions* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The following table presents the additional attributes and/or associations for *SDMNDefinitions*:

Table 8. SDMNDefinitions Attributes and/or Associations

Property/Association	Description
dataItem : DataItem [0..*]	This is a list of the **DataItems** that are included in the *SDMNDefinitions*. See the section entitled "DataItems," below, for more information about DataItems.
dataState : DataState [0..*]	This is a list of the potential *DataStates* that can be associated with a **DataItem**.
itemFormat : ItemFormat [0..*]	A list of potential *ItemFormats* for **DataItems**. This will apply mainly to electronic documents (such as .pdf).
itemDefinition : ItemDefinition [0..*]	This is a list of the ItemDefinitions that are included in the *SDMNDefinitions*. See the section entitled "Item Definitions," below, for more information about ItemDefinitions.
location : Location [0..*]	A list of potential *Locations* for **DataItems**.
sharedDataModel : SharedDataModel [0..*]	This is a list of SharedDataModels that included in the *SDMNDefinitions*. See the section entitled "SharedDataModel," below, for more information.

9.2 SDMN Vocabularies

Vocabularies (lists of terms) can be added to an *SDMNModel*. *SDMNVocabularies* are sets of terms that can be defined by an external ontology. The terms link to formal definitions for the model elements that are created by the modeling language. The *SemanticReference* element is used to name the term and provide a link to the definitions. These terms/definitions can then be associated with the appropriate model elements. *SDMNVocabularies* are contained within an *SDMNModel*.

The figure below displays the *SDMNVocabulary* metamodel (including the two predefined instances for *SDMNVocabulary*):

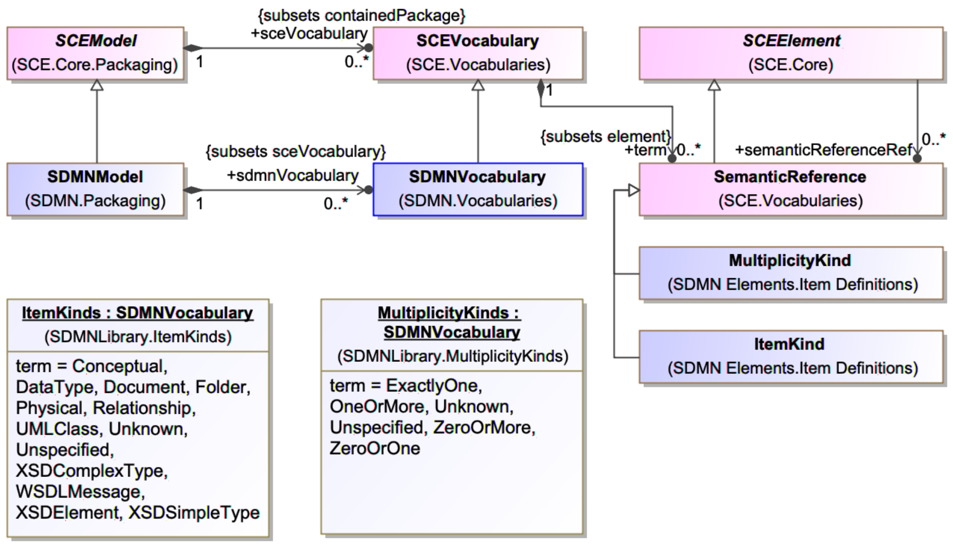

SCEModel
(SCE.Core.Packaging)
{subsets containedPackage}
+sceVocabulary
1
0..*
SCEVocabulary
(SCE.Vocabularies)
SCEElement
(SCE.Core)
1
{subsets element}
+term
0..*
+semanticReferenceRef
0..*
{subsets sceVocabulary}
+sdmnVocabulary
SDMNModel
(SDMN.Packaging)
1
0..*
SDMNVocabulary
(SDMN.Vocabularies)
SemanticReference
(SCE.Vocabularies)
MultiplicityKind
(SDMN Elements.Item Definitions)
ItemKind
(SDMN Elements.Item Definitions)
ItemKinds : SDMNVocabulary
(SDMNLibrary.ItemKinds)
term = Conceptual, DataType, Document, Folder, Physical, Relationship, UMLClass, Unknown, Unspecified, XSDComplexType, WSDLMessage, XSDElement, XSDSimpleType
MultiplicityKinds : SDMNVocabulary
(SDMNLibrary.MultiplicityKinds)
term = ExactlyOne, OneOrMore, Unknown, Unspecified, ZeroOrMore, ZeroOrOne

Figure 11: The SDMNVocabulary Metamodel

9.2.1 SDMNVocabulary

An *SDMNVocabulary* is a list of terms, through the *SemanticReference* element, that can be used to relate to model elements to the external definition or meaning. The terms themselves do not represent the definitions or meanings but provide links to an external source. Multiple *SDMNVocabularies* can be defined. They are contained in an *SDMNModel.*

Further, *SDMNVocabularies* can be used for creating a user-defined list of enumerated values for use within a **SDMN** (as opposed to a fixed enumeration list). It is up to the **SDMN** modeling tool to organize the *SDMNVocabularies* into the appropriate enumerated lists. Since the *SemanticReference* element has a name and the links to external definitions are optional, the list (the "enumeration" *SDMNVocabularies*) can be created before the specific external definitions are established.

SDMN has two pre-defined *SDMNVocabularies* for the enumerated terms for the *ItemKind* element (see the section entitled "ItemKind" for more information) and the *MultiplicityKind* (see the section entitled "MultiplicityKind" for more information).

Generalizations

The *SDMNVocabulary* element inherits the attributes and/or associations of:

- *SCEVocabulary* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The *SDMNVocabulary* element does not have any additional attributes and/or associations.

10 SDMN Model Elements

This chapter defines **DataItem** and its related elements and *ItemDefinition* and its related elements.

10.1 DataItems

Several elements in **BPM+** Models are intended to store or convey data required for the execution of those Models. **BPMN** has Data Objects, Data Inputs, Data Outputs, Data Stores, and Properties. **CMMN** has Case File Items. **DMN** has Information Items that are used for Data Inputs and Decisions. While there are some technical differences between how data is structured and used across the **BPM+** specifications, at the logical level, they all play the same role within the respective languages. This is evident when a specific conceptual data element (e.g., "Invoice") can be included in all three **BPM+** modeling languages. That is, the same data element can be passed from a **CMMN** Case to a **BPMN** Process and then used in a **DMN** Decision. Currently, the same logical data element has to be defined separately in each modeling language. If there are a lot of data elements that are shared between the models of a BPM+ Knowledge Package, the development and maintenance burden for synchronizing the properties of the data elements will problematic. This is the driver for defining a **DataItem**.

Thus, a **SDMN DataItem** represents a common definition and structure for the data handling elements of the other **BPM+** models.

A **DataItem** 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 **DataItem** can be anything from a folder or document stored with CMIS, an entire folder hierarchy referring or containing other **DataItems**, 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 *ItemDefinition* (see below). This may include the definition of properties ("metadata") of a **DataItem**, which is only applied to **CMMN** Case File Items. If the internal content of the **DataItem** is known, an XML Schema, describing the **DataItem**, may be imported.

To support **CMMN** CaseFileItems, **DataItems** can be organized into arbitrary hierarchies either by containment or by reference.

For containment hierarchies the associations children and parent are used whereas for reference hierarchies the associations `targetRefs` and `sourceRef` are used. 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.

Several elements in **BPMN** are subject to store or convey items during process execution. These elements are referenced generally as "item-aware elements." This is similar to the variable construct common to many languages. As with variables, these elements have an ItemDefinition.

The data structure these elements hold is specified using an associated *ItemDefinition*. An ItemAwareElement MAY be underspecified, meaning that the structure attribute of its ItemDefinition is optional if the modeler does not wish to define the structure of the associated data. The elements in the specification defined as item-aware elements are: Data Objects, Data Object References, Data Stores, Properties, DataInputs and DataOutputs.

The following figure shows the metamodel elements related to the **DataItem** element.

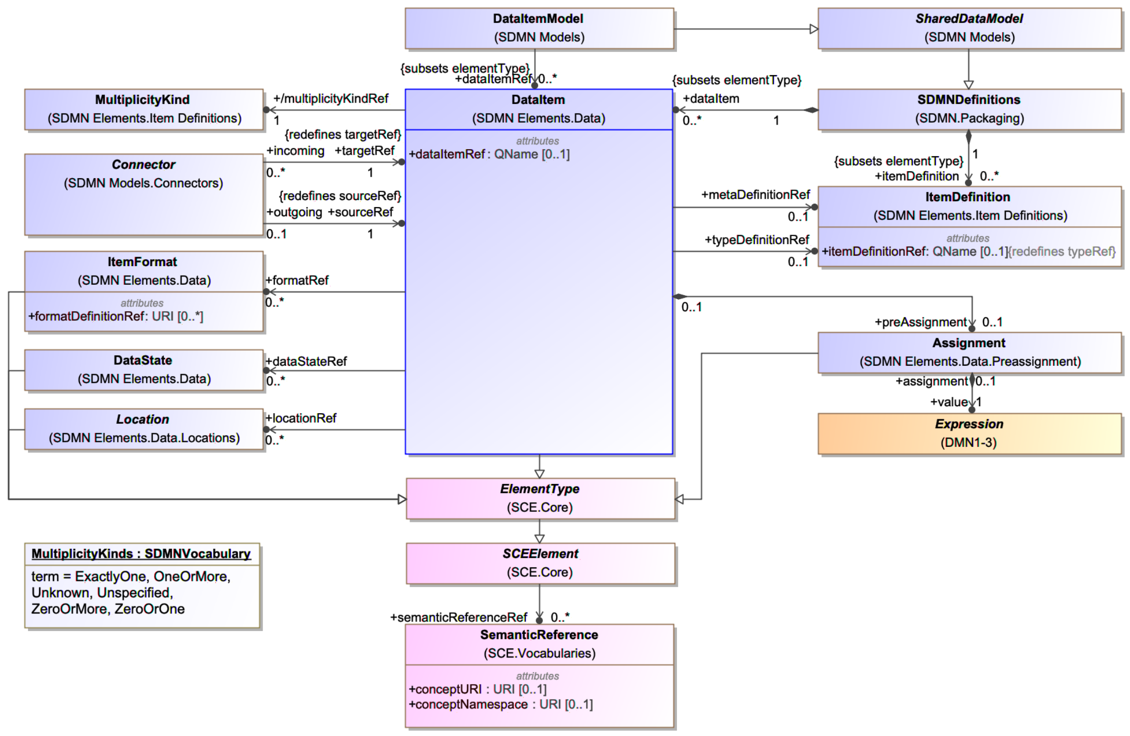


Figure 12: The DataItem Metamodel

10.1.1 DataItem

A **SDMN DataItem** represents a common definition and structure for the data handling elements of the other **BPM+** models (as described above). It is contained within a *SDMNDefinitions*.

Notation

The following statements define the notation for a **DataItem**:

- A **DataItem** is a shape that SHALL be a document shape with folded upper right corner and drawn with a single line (see below).

The use of text, color, size, and lines for a **Reference Connector** SHALL follow the rules defined in the section entitled "

  - Use of Text, Color, Size, and Lines in a Diagram" above.

The **DataItem** shape is a document with folded upper right corner (see figure below). This is the default notation for a **DataItem** and occurs when the `itemKind` property of the *ItemDefintion* assigned to the **DataItem** is set to anything other than `folder`, which has a different notation as shown below.

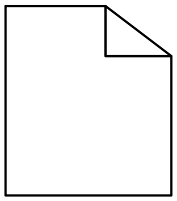


Figure 13: A DataItem Object

The **DataItem** shape is a folder when the `itemKind` property of the *ItemDefintion* assigned to the **DataItem** is set to `folder`. (see figure below).

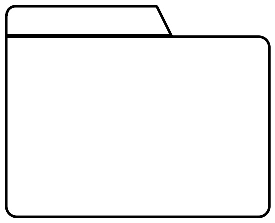


Figure 14: A DataItem Object

Generalizations

The **DataItem** element inherits the attributes and/or associations of:

- *ElementType* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The following table presents the additional attributes and/or associations for **DataItem**:

Table 9. DataItem Attributes and/or Associations

Property/Association	Description
dataItemRef : QName [0..1]	A reference to an external **DataItem** that is imported into this Shared Data Model. The **DataItem** and its details can only be viewed in this model. Any changes to the original SHALL be carried out in the source Shared Data Model. A **DataItem** can have only one of `dataItemRef` or `ItemDefinitionRef` as a set attribute. Neither of them is required, though. If a `dataItemRef` is defined, then the graphical notation for the **DataItem** will include a locked icon.

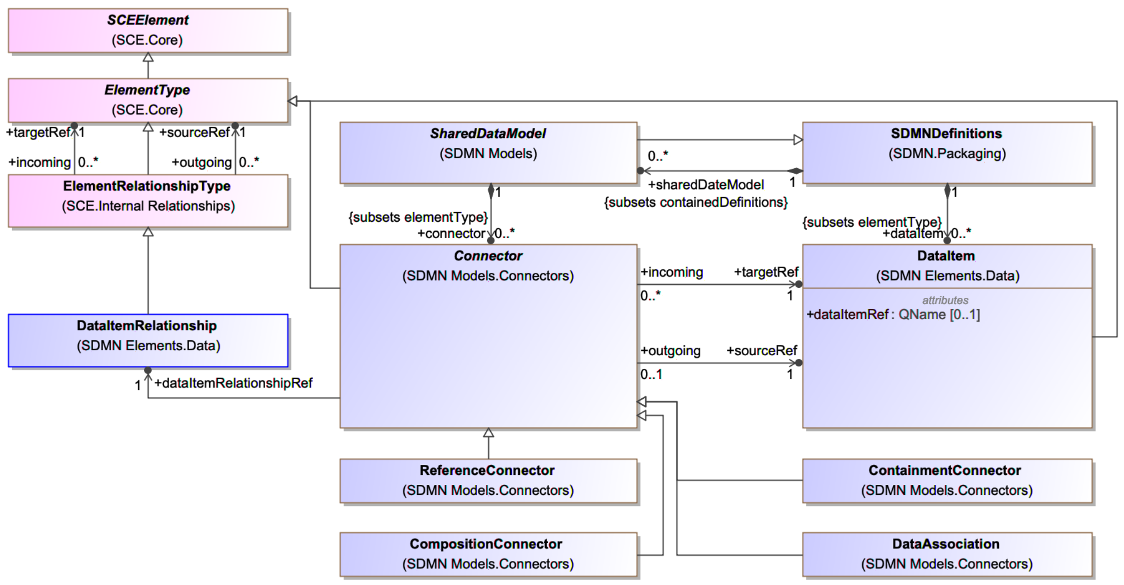
dataStateRef : DataState [0..*]	A **DataItem** can have multiple *DataStates*, which represent significant states in its lifecycle. The *DataStates* of a **DataItem** may show up as Milestones within a CMMN Case.
formatRef : ItemFormat [0..1]	A list of potential *ItemFormats* for the **DataItem**. This will apply mainly to electronic documents (such as .pdf).
locationRef : Location [0..*]	A list of potential *Locations* for the **DataItem**.
metaDefinitionRef : ItemDefinition [0..1]	A reference to an *itemDefinition* that defines the *Properties* of the **DataItem**. The `itemComponents` of the *ItemDefinition* structure map to the Properties of a **CMMN** Case File Item. Each of the `itemComponents` SHALL be a simple type.
multiplicityKindRef : MultiplicityKind [1] default: ExactlyOne	This attribute sets the multipliciy of the **DataItem**. The default is `ExactlyOne`. This attribute SHALL have the same value as the `multiplicty` attribute of the associated *ItemDefinition*, if there is one. If the `mutiplicity` is set to `ZeroOrMore`, or `OneOrMore` then the graphical notation for the **DataItem** will include a **Collection** (multi-instance) icon.
preAssignment : Assignment [0..1]	Specifies an optional pre-assignment **DMN** *Expression*. The expression will provide values for one or more of the simple type `itemComponents` of the *ItemDefinition* set for the **DataItem**.
typeDefinitionRef : ItemDefinition [0..1]	A reference to an *itemDefinition* that defines the detailed structure, which can be simple or complex, of the **DataItem**. A **DataItem** can have only one of `dataItemRef`, or `typeDefinitionRef` as a set attribute. None of them are required, though.

10.1.2 DataItemRelationship

An *DataItemRelationship* is used to define a relationship between **DataItems**. This relationship will specify that one **DataItem** is connected in some way to another **DataItem** – there is some type of relationship. This relationship is included to support the source and target reference associations that exists between **CMMN** CaseFileItems. For example, the CaseFileItems might be created at the same time (although independently) or they are often sent together during the performance of a **CMMN** Case, etc. In a sense, it is a non-graphical way of grouping **DataItems**.

They are contained in the *SDMNDefinitions* package.

The following figure shows the metamodel elements related to the *DataItemRelationship* element.



SCEElement (SCE.Core)
ElementType (SCE.Core)
+targetRef 1
+sourceRef 1
+incoming 0..*
+outgoing 0..*
ElementRelationshipType (SCE.Internal Relationships)
DataItemRelationship (SDMN Elements.Data)
1 +dataItemRelationshipRef
SharedDataModel (SDMN Models)
SDMNDefinitions (SDMN.Packaging)
0..*
+sharedDateModel 1
{subsets containedDefinitions}
{subsets elementType}
+connector 0..*
{subsets elementType}
+dataItem 0..*
Connector (SDMN Models.Connectors)
+incoming +targetRef
0..* 1
+outgoing +sourceRef
0..1 1
DataItem (SDMN Elements.Data)
attributes
+dataItemRef : QName [0..1]
ReferenceConnector (SDMN Models.Connectors)
ContainmentConnector (SDMN Models.Connectors)
CompositionConnector (SDMN Models.Connectors)
DataAssociation (SDMN Models.Connectors)

Figure 15: The DataItemRelationship Metamodel

Generalizations

The *DataItemRelationship* element inherits the attributes and/or associations of:

- *ElementRelationshipType* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The following table presents the additional attributes and/or associations for *ReferenceRelationship*:

Table 10. ReferenceRelationship Attributes and/or Associations

Property/Association	Description
sourceRef : DataItem [1]	The source **DataItem**. For reference hierarchies of a **DataItem**, `source` refers to the source of the **DataItem**. If **DataItem** b is a `target` of **DataItem** a, then `source` of **DataItem** b is a. This redefines the `sourceRef` association inherited from **SCE**:ElementRelationshipType.
targetRef : DataItem [1]	The target **DataItem**. The set of *DataItemRelationship* `targets` of a **DataItem** MUST NOT refer that **DataItem** or any **DataItem** in which that **DataItem** is referred. This avoids cycles in the references. This redefines the `targetRef` association inherited from **SCE**:ElementRelationshipType.

10.1.3 DataState

DataItems can optionally reference a *DataState* element, which is the state of the data contained in the **DataItem**. The definition of these *DataStates*, e.g., possible values and any specific semantic are out of scope of this specification. Therefore, **SDMN** adopters can use the *DataState* element and the **SDMN** extensibility capabilities to define their *DataStates*.

Generalizations

The *DataState* element inherits the attributes and/or associations of:

- *ElementType* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The *DataState* element does not have any additional attributes and/or associations.

10.1.4 ItemFormat

Represents the format of an **DataItem**. It can be something as simple as "mime types" or the specification of a format documented in a formal format registry. *ItemFormats* are contained within a *SDMNDefinitions* and can be referenced by **DataItems**.

Generalizations

The *ItemFormat* element inherits the attributes and/or associations of:

- *ElementType* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The following table presents the additional attributes and/or associations for *ItemFormat*:

Table 11. ItemFormat Attributes and/or Associations

Property/Association	Description
formatDefinitionRef : URI [0..*]	The identifier of the format within the specified format registry. For example "dicom" if the registry is that of W3C mime types. This is not the usual "id" found commonly in this specification. This is a "stringified" (if necessary) unique id in the context of the .formatRegistry.

10.1.5 Locations

The Locations package contains elements related to physical or virtual locations. Organizations may deem the locations at which a **DataItem** may exist to be of significance. *Locations* are often tracked in the context of pedigree and provenance.

10.1.5.1 Location

Location is an abstract class where its concrete specializations identify a particular place or position. *Locations* are contained within a *SDMNDefinitions* and can be referenced by **DataItems**.

The following figure shows the metamodel elements related to the *Location* element.

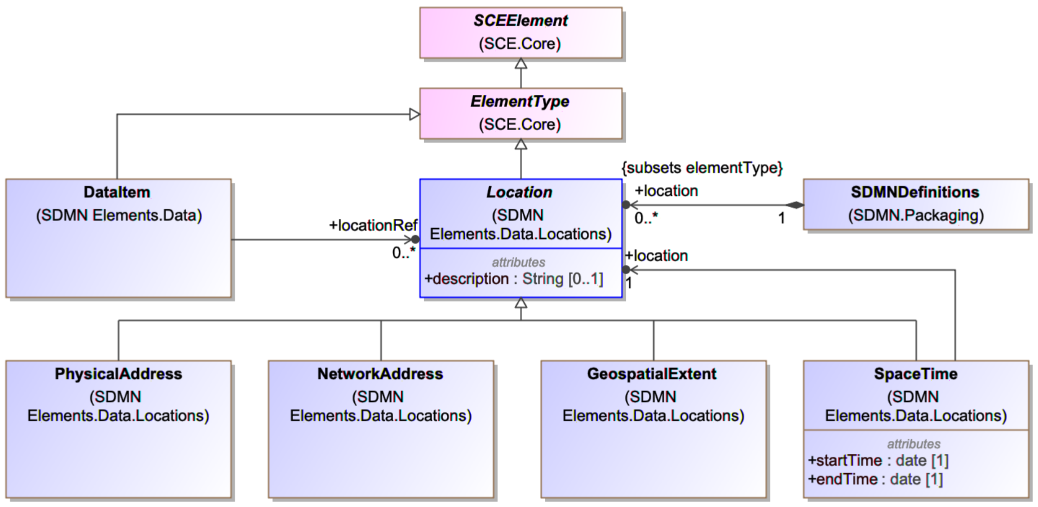


Figure 16: The Location Metamodel

Generalizations

The *Location* element inherits the attributes and/or associations of:

- *ElementType* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The following table presents the additional attributes and/or associations for *Location*:

Table 12. Location Attributes and/or Associations

Property/Association	Description
description : String [0..1]	A description of the *Location*.

10.1.5.2 GeospatialExtent

A location that is a volume in the world such as a container or a room.

Generalizations

The *GeospatialExtent* element inherits the attributes and/or associations of:

- *Location* (see the section entitled "Location" for more information).

Further, the *Location* element inherits the attributes and/or associations of:

- *ElementType* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The *GeospatialExtent* element does not have any additional attributes and/or associations.

10.1.5.3 NetworkAddress

The address of an element or node on a network.

Generalizations

The *NetworkAddress* element inherits the attributes and/or associations of:

- *Location* (see the section entitled "Location" for more information).

Further, the *Location* element inherits the attributes and/or associations of:

- *ElementType* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The *NetworkAddress* element does not have any additional attributes and/or associations.

10.1.5.4 PhysicalAddress

A physical location in the real world.

Generalizations

The *PhysicalAddress* element inherits the attributes and/or associations of:

- *Location* (see the section entitled "Location" for more information).

Further, the *Location* element inherits the attributes and/or associations of:

- *ElementType* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The *PhysicalAddress* element does not have any additional attributes and/or associations.

10.1.5.5 SpaceTime

A *Location* at a particular point in time.

Generalizations

The *SpaceTime* element inherits the attributes and/or associations of:

- *Location* (see the section entitled "Location" for more information).

Further, the *Location* element inherits the attributes and/or associations of:

- *ElementType* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The following table presents the additional attributes and/or associations for *SpaceTime*:

Table 13. SpaceTime Attributes and/or Associations

Property/Association	Description
endTime : Date [1]	The ending time of the *SpaceTime*.

location : Location [1]	The location of the *SpaceTime*.
startTime : Date [1]	The starting time of the *SpaceTime*.

10.1.6 Pre-Assigning Values for DataItems

There are situations in the development of a BPM+ Knowledge Package when the values of some of the **DataItem** properties are known. For example, in a healthcare scenario, certain medications are recommended for a particular condition. Each medication with have a representative **DataItem** in the **Shared Data Model** that will share the same *ItemDefintion*. The *ItemDefintion* will define the properties that are needed for prescribing the medication, such as medication name, codes, dosages, etc.

Assignment

An Assignment is contained within a **DataItem** or a **DataAssociation**.

Generalizations

The *Assignment* element inherits the attributes and/or associations of:

- *ElementType* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The following table presents the additional attributes and/or associations for *Assignment*:

Table 14. Assignment Attributes and/or Associations

Property/Association	Description
value : Expression [1]	The **DMN** *Expression* that evaluates the *Assignment*.

Pre-Assignment Example

The following example is a FEEL expression that preassigns values for a “medication” **DataItem**.

```
{
   "Metoprolol Tartrate 25" : {
      Id : "metoprololTartrate25Medication" ,
         Code: {
         Coding : {
            System : "http://www.nlm.nih.gov/research/umls/rxnorm" ,
            Code : "866426"
         },
         Text : "Metoprolol Tartrate 25 MG"
      },
      Form : {
         Coding : {
            System : "http://snomed.info/sct" ,
            Code : "385055001" ,
            Display : "Tablet dose form"
         },
         Text : "Tablet dose form"
      },
      Ingrediant : {
            Substance : {
                Id : "metoprololTartrate25Substance" ,
                Code : {
                    Coding : {
                        System : "http://www.nlm.nih.gov/research/umls/rxnorm" ,
                        Code : "6918"
                    }
                    Text : "Metoprolol"
                }
            }
            Strength : {
                Numerator : {
                    Value : "25" ,
                    Unit : "mg"
                },
                Denominator : {
                    Value : "1" ,
                    Unit : "{tbl}"
                }
            }
        }
    }
}
```

10.2 Item Definitions

The *ItemDefinition* element is the mechanism for providing the data structure of **DataItems**.

The following figure shows the metamodel elements related to the *ItemDefinition* element.

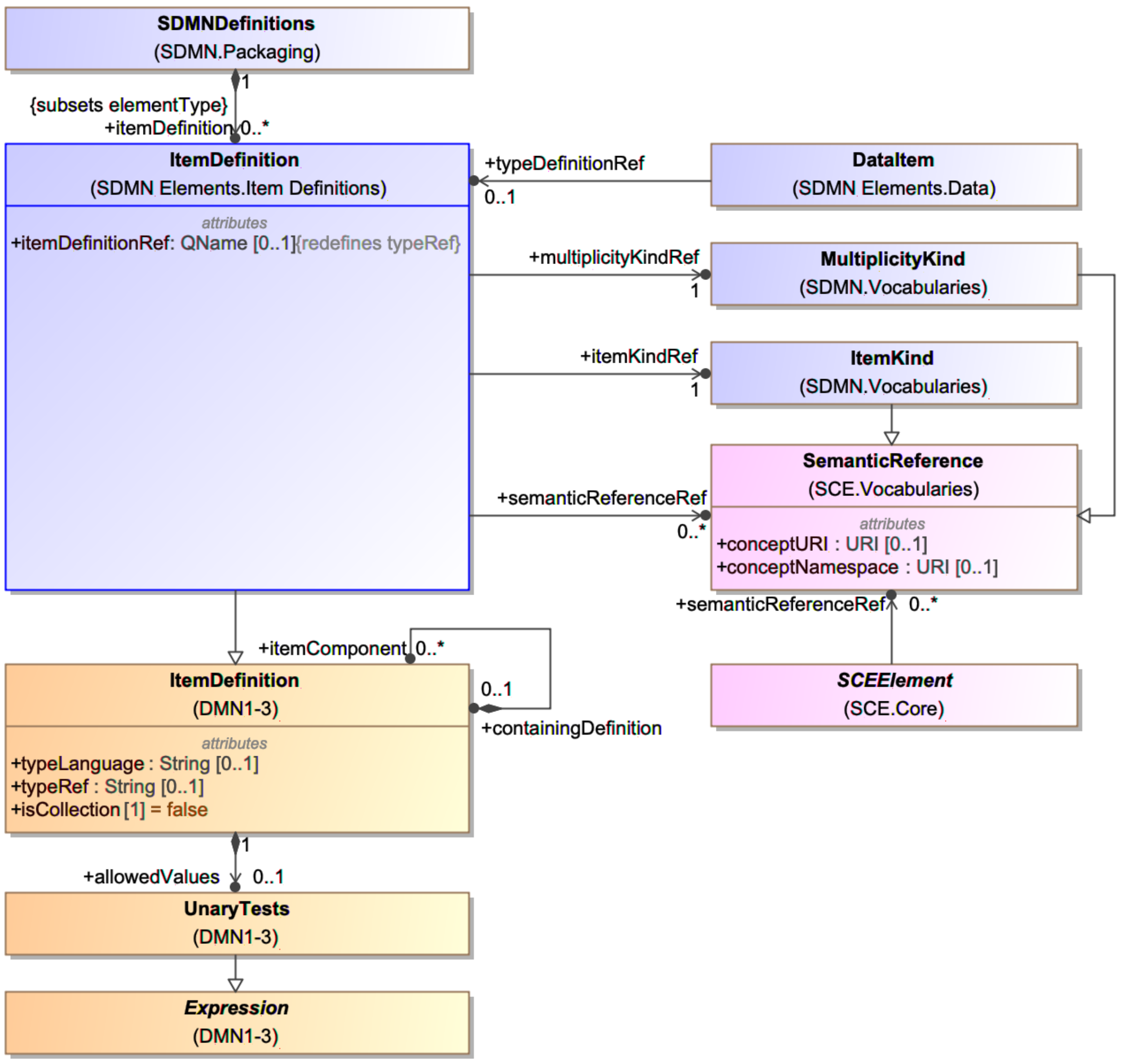


Figure 17: The ItemDefinition Metamodel

10.2.1 ItemDefinition

The *itemDefinition* element is the mechanism for providing the data structure of **DataItems**. It is contained within a *SDMNDefinitions.*

Generalizations

The *ItemDefinition* element inherits the attributes and/or associations of:

- **DMN** *ItemDefinition* (see the **DMN** specification for more information [OMG doc number formal-2021-01-01]).

Properties

The following table presents the additional attributes and/or associations for *ItemDefinition*:

Table 15. ItemDefinition Attributes and/or Associations

Property/Association	Description
itemDefinitionRef : QName [0..1]	A reference to an external *ItemDefinition* that is imported into this **Shared Data Model**. The *ItemDefinition* and its details can only be viewed in this model. Any changes to the original SHALL be carried out in the source **Shared Data Model**. Other types of structures are not allowed for the **SDMN**. However, **BPMN** Data Objects and **CMMN** Case File Items have the capability of references other types of structures. These other types of structures would not be a part of the **SDMN Shared Data Model**.
itemKindRef : ItemKind [1]	This defines the nature of the **DataItem**. Possible values are physical, information, conceptual, and others (see the section entitled "ItemKind." The default value is `information`. If the *ItemDefinition* has `itemComponents` or `itemComponentRefs`, then the `itemKind` for each of these sub-*ItemDefinitions* SHALL match the top-level *ItemDefinition*.
multiplicityKindRef : MultiplicityKind [1] default: ExactlyOne	This sets the multipliciy of the *ItemDefinition*. The default is `ExactlyOne`. This attribute SHALL have the same value as the `multiplicty` attribute of the associated **DataItem**. This attribute redefines the isCollection attribute of the **DMN** ItemDefinition.
semanticReferenceRef : (SCE) SemanticReference [0..*]	A *ItemDefinition* can include multiple **SCE** *SemanticReference* elements. This attribute was added because *ItemDefinition* is based on the **DMN** ItemDefinition, which is not based on the **SCE** specification and thus, does not have a built in *SemanticReference* as part of its definition. See the section entitled "Semantic Reference" in the **SCE** specification for more information.

10.2.2 ItemKind

This class is a type of *SemanticReference* that serves as the `terms` for an *SDMNVocabulary* that is used to specific the kind of multiplicity that exists for an *ItemDefintion*. Instead of being defined a fixed enumerated list, the kinds can be defined through a class (*ItemKind*) and instances of that class (as shown below). The instances defined in the **SDMN** Library SHALL be included in any **SDMN** implementation. However, the implementation can allow additional instances of this class if required for a particular modeling situation (see the section entitled "MultiplicityKinds" for more information). Some of the literals for *ItemKind* are based on the **CMMN** CaseFileItemDefinition literals for `DefinitionType`.

In practice, when a modeler creates a model with an *ItemDefintion*, the *ItemKind* will be instantiated by one of the 13 instances in the Library. The *ItemKind* identifies the different natures that *ItemDefinitions*, and thus, the **DataItems** that refence the *ItemDefinitions*, may represent.

The following figure shows the metamodel elements related to the *ItemKind* element (which includes the standard set of instances provided by the **SDMN** Library).

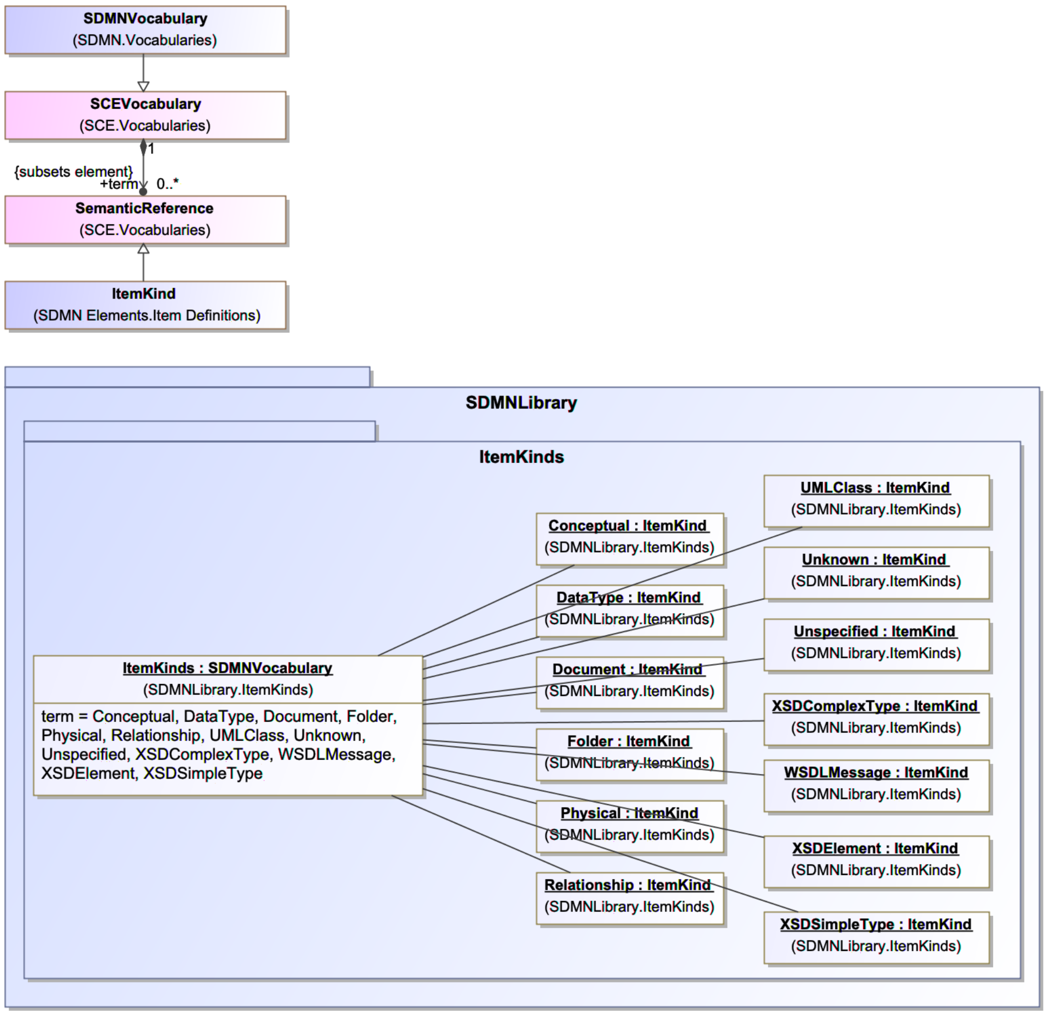


Figure 18: The ItemKind Metamodel

Generalizations

The *ItemKind* element inherits the attributes and/or associations of:

- *SemanticReference* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The *ItemKind* element does not have any additional attributes and/or associations.

Standard Terms Vocabulary

The following table presents a description for the included instances for *ItemKind*:

Table 16. ItemKind Literals

Literal	Description
Conceptual	The type of the *ItemDefinition* that doesn't represent data or physical items, but represents concepts in the minds of users that are important for tasks or decisions. For example, a preference for a particular type of procudure will influence a doctor's decision. While actual computations cannot be made with `Conceptual` *ItemDefinitions*, they are used to document aspects of the modeled behaviors.
DataType	The type of the ItemDefinition that fully utilizes the structural data capabilities inherent to ItemDefinition. Using FEEL, these will define simple types or data structures.
Document	This represents a Data Object or Case File Item that is a type of `Document`. In **BPMN**, the document could be physical (e.g., printed) or electronic. In **CMMN**, it would represent a document in a Document Management System and is defined through the following URI: http://www.omg.org/spec/CMMN/DefinitionType/CMISDocument
Folder	This represents a **CMMN** Case File Item that is a `Folder`. A `Folder` can contain other `Folders` or `Documents`. These relationships are set through the `Child` and `Parent` attributes of the **DataItem**. Neither **BPMN** nor **DMN** have the concept of `Folder` as a data element. Thus, **DataItems** based on a `Folder` *ItemDefinition* would not map to **BPMN** or **DMN** data elements. The `Folder` is defined through the following URI: http://www.omg.org/spec/CMMN/DefinitionType/CMISFolder
Physical	The *ItemKind* is represents objects in a **BPMN** Process that are physical objects, such as printed documents or manufactured items. These types of **DataItems** are not currently relevant to **CMMN** or **DMN**.
Relationship	The *ItemKind* is defined through the following URI: http://www.omg.org/spec/CMMN/DefinitionType/ CMISRelationship
UMLClass	The *ItemKind* is represents a **UML** Class in a Class Diagram.
Unknown	The *ItemKind* is defined through the following URI: http://www.omg.org/spec/CMMN/DefinitionType/Unknown
Unspecified	The *ItemKind* is defined through the following URI: http://www.omg.org/spec/CMMN/DefinitionType/Unspecified
WSDLMessage	The *ItemKind* is represents a **WSDL** Message.
XSDComplexType	For *ItemKinds* of this type, the (**SCE**) *Import* class SHOULD be used to import an XML Schema definition into the **Shared Data Model**. The *ItemKind* is defined through the following URI: http://www.omg.org/spec/CMMN/DefinitionType/ XSDComplexType

XSDElement	For *ItemKinds* of this type, the (**SCE**) *Import* class SHOULD be used to import an XML Schema definition into the **Shared Data Model**. The *ItemKind* is defined through the following URI: http://www.omg.org/spec/CMMN/DefinitionType/XSDElement
XSDSimpleType	For *ItemKinds* of this type, the (**SCE**) *Import* class SHOULD be used to import an XML Schema definition into the **Shared Data Model**. The *ItemKind* is defined through the following URI: http://www.omg.org/spec/CMMN/DefinitionType/XSDSimpleType

10.2.3 MultiplicityKind

This class is a type of *SemanticReference* that serves as the `terms` for an *SDMNVocabulary* that is used to specific the kind of multiplicity that exists for an *ItemDefintion* and a **DataItem**. Instead of being defined a fixed enumerated list, the kinds can be defined through a class (*MultiplicityKind*) and instances of that class (as shown below). The instances defined in the **SDMN** Library SHALL be included in any **SDMN** implementation. However, the implementation can allow additional instances of this class if required for a particular modeling situation (see the section entitled "MultiplicityKinds" for more information).

In practice, when a modeler creates a model with an *ItemDefintion* and a **DataItem**, the *MultiplicyKind* will be instantiated by one of the six instances in the Library. This set of instances is based on the **CMMN** CaseFileItem multiplicity setting. These kinds can be mapped to the **BPMN** Collection setting and the **DMN** Collection setting. See the section below for the mappings.

The following figure shows the metamodel elements related to the *MultiplicityKind* element (which includes the standard set of instances provided by the **SDMN** Library).

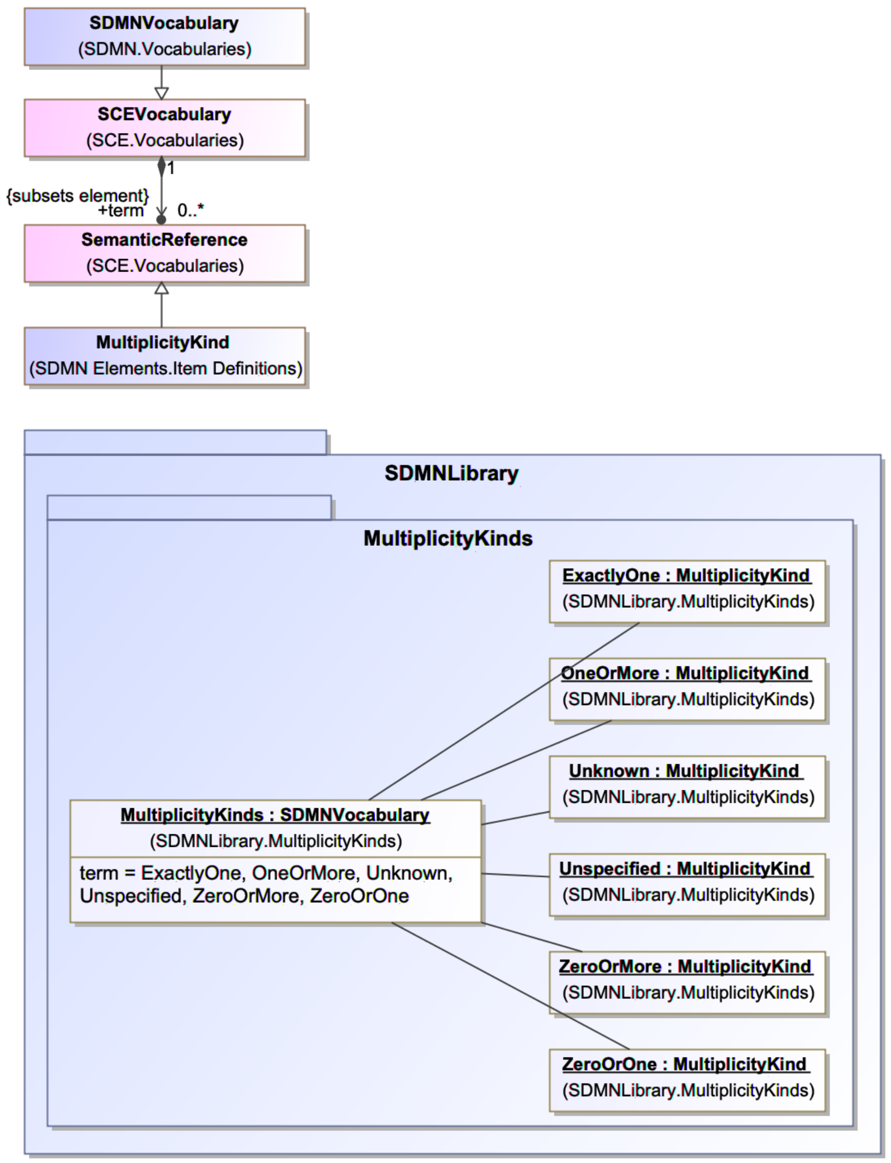

SDMNVocabulary
(SDMN.Vocabularies)
SCEVocabulary
(SCE.Vocabularies)
1
{subsets element}
+term
0..*
SemanticReference
(SCE.Vocabularies)
MultiplicityKind
(SDMN Elements.Item Definitions)
SDMNLibrary
MultiplicityKinds
ExactlyOne : MultiplicityKind
(SDMNLibrary.MultiplicityKinds)
OneOrMore : MultiplicityKind
(SDMNLibrary.MultiplicityKinds)
Unknown : MultiplicityKind
(SDMNLibrary.MultiplicityKinds)
MultiplicityKinds : SDMNVocabulary
(SDMNLibrary.MultiplicityKinds)
term = ExactlyOne, OneOrMore, Unknown, Unspecified, ZeroOrMore, ZeroOrOne
Unspecified : MultiplicityKind
(SDMNLibrary.MultiplicityKinds)
ZeroOrMore : MultiplicityKind
(SDMNLibrary.MultiplicityKinds)
ZeroOrOne : MultiplicityKind
(SDMNLibrary.MultiplicityKinds)

Figure 19: The MultiplicyKind Metamodel

Generalizations

The *MultiplicityKind* element inherits the attributes and/or associations of:

- *SemanticReference* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The *MultiplicityKind* element does not have any additional attributes and/or associations.

Standard Terms Vocabulary

The following table presents a description for the included instances for *MultiplicityKind*:

Table 17. MultiplicityKind Literals

Literal	Description
ExactlyOne	There is one copy of this *ItemDefintion* or **DataItem**.
OneOrMore	There is at least one copy of this *ItemDefintion* or **DataItem**, but there may be more.
Unknown	The muliplicty is not know for this *ItemDefintion* or **DataItem**.
Unspecified	The muliplicty is not specified for this *ItemDefintion* or **DataItem**.
ZeroOrMore	There may be no copies of this *ItemDefintion* or **DataItem** or there may be multiple copies.
ZeroOrOne	There may be no copies of this *ItemDefintion* or **DataItem** or there may be one copy.

11 SDMN Models

The main purpose of **SDMN** is to allow modelers to create **DataItem** models, through a diagramming tool, to define the elements that are required for other BPM+ models, such as **BPMN**. This chapter defines the elements for constructing such models.

11.1 SharedDataModel

SharedDataModel is the abstract element that provides the foundation for the concrete **SDMN** models. Currently, there is only one concrete model, **DataItemModel** (see below). In future versions of **SDMN**, addition models can be added. It is a type of *SDMNDefinitions* and is contained in an *SDMNDefinitions*.

The following figure shows the metamodel elements related to the *SharedDataModel* element.

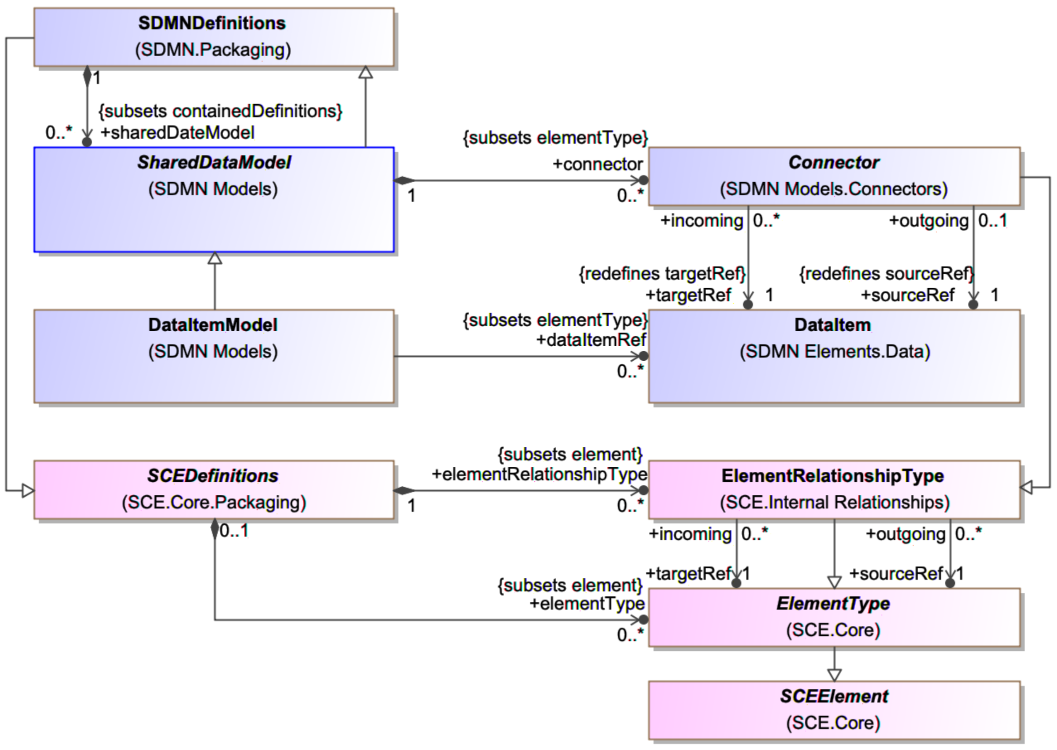


Figure 20: The SharedDataModel Metamodel

Generalizations

The *SharedDataModel* element inherits the attributes and/or associations of:

- *SCEDefinitions* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The following table presents the additional attributes and/or associations for *SharedDataModel*:

Table 18. SharedDataModel Attributes and/or Associations

Property/Association	Description
connector : Connector [0..*]	This is a list of the *Connectors* (**Composition**, **Containment**, **Reference**, and **Data Association**) that are included in the *SharedDataModel*. See the section entitled "Connectors," below, for more information about *Connectors*.

11.2 DataItemModel

The **DataItemModel** is mechanism for creating **SDMN** models. Through a modeling tool, which provides the specified notation, these models can be created to support other BPM+ languages, particularly in the context of a BPM+ Knowledge Package. It is contained in an *SDMNDefinitions*.

The figure below displays an example of a **DataItemModel** for the Hello Patient use case.

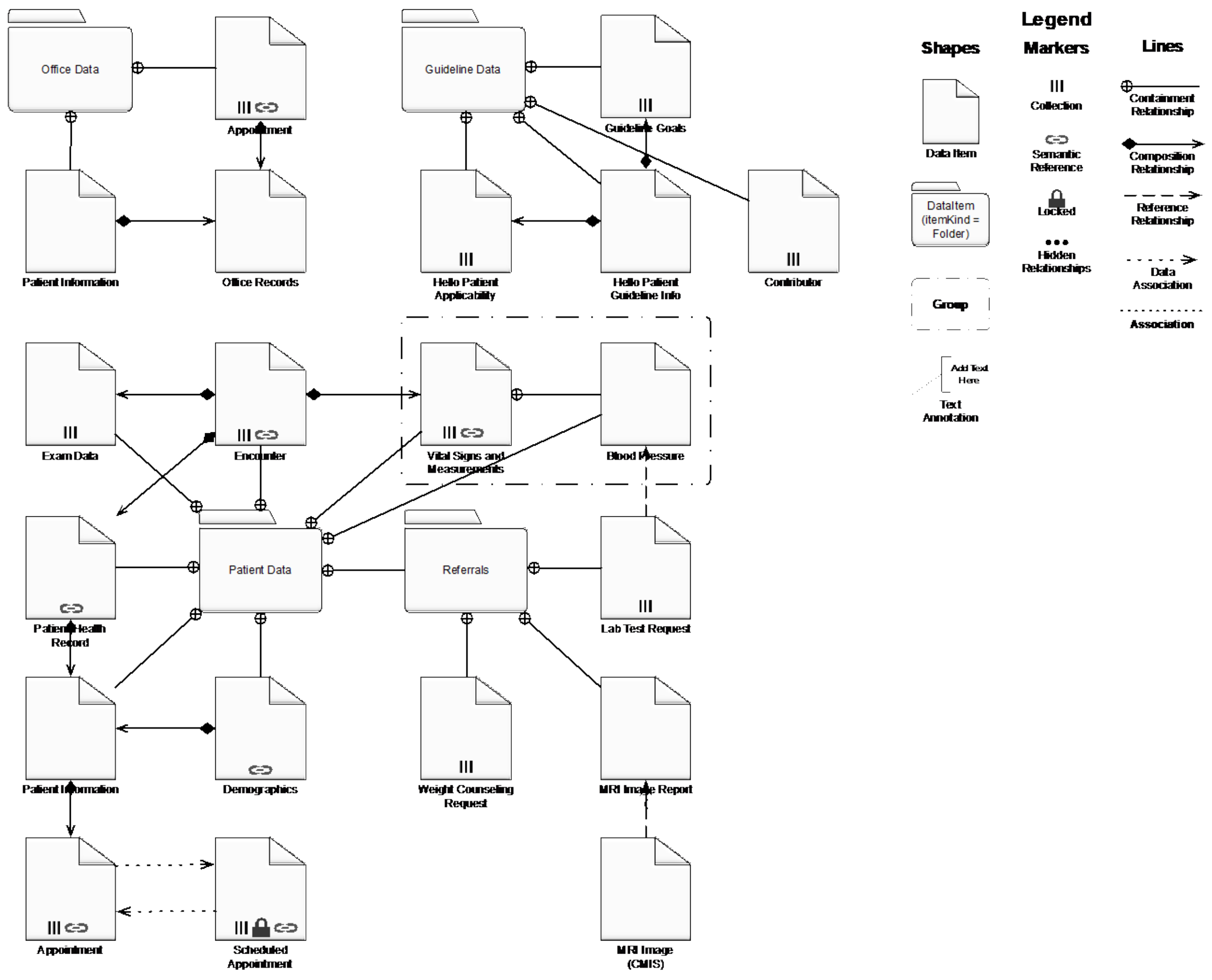


Figure 21: Example of the "Hello Patient" DataItem Model

Generalizations

The **DataItemModel** element inherits the attributes and/or associations of:

- *SDMNModel* (see the section entitled "SDMNModel" for more information).

Further, the *SDMNModel* element inherits the attributes and/or associations of:

- *SCEModel* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The following table presents the additional attributes and/or associations for **DataItemModel**:

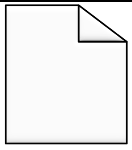
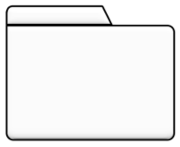
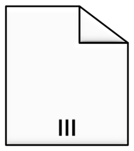
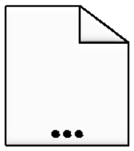
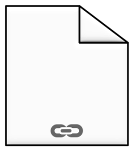
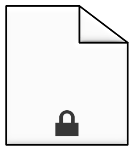
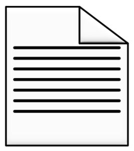
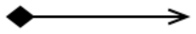
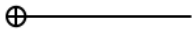
Table 19. DataItemModel Attributes and/or Associations

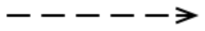
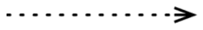
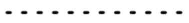
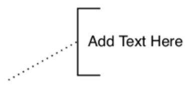
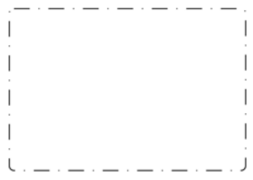
Property/Association	Description
dataItemRef : DataItem [0..*]	This is a list of the **DataItems** that are in the **DataItemModel**.

11.2.1 Graphical Elements

The table below displays the graphical elements of a **DataItem Model**:

Table 20. Shared Data Model Graphical Elements

Element	Description	Notation
DataItem	This is a **DataItem** that is set to a any type, except `folder`, through the `itemKind` property of the *ItemDefintion* assigned to the **DataItem**. A document shape with folded upper right corner is default notation for a **DataItem** (see figure to the right).	
Parent DataItem (folder)	This is a **DataItem** that is set to a `folder` type through the `itemKind` property of the *ItemDefintion* assigned to the **DataItem**. For this variation of **DataItem**, its notation is set to a folder shape (see figure to the right).	
DataItem (Collection)	Note that this marker can be used in combination with any of the following four markers shown in this table.	
DataItem (with Hidden Relationships – children)		
DataItem with Semantic Reference		
Locked DataItem		
DataItem with pre-assigned data	The lines added within the **DataItem** shape indicate that some of the **DataItem** properties have been set with pre-assigned values through a *LiteralExpression*.	
Composition Connector	This is a *Connector* that represents a composition relationship between two **DataItems**.	
Containment Connector	This is a *Connector* that represents a containment relationship between two **DataItems**.	

Reference Connector	This is a *Connector* that represents the existence of a relationship between two **DataItems**. The nature of that relationship is at the descretion of the modeler.	
Data Association	This is a *Connector* that represents the existence of mappings and/or transformations of data structure elements between the two **DataItems**.	
Association (see the **SCE** specification)	The **Association** connector, defined in **SCE** [OMG doc number bmi-2021-12-09], allows a Shared Data Model developer to connect two objects in the diagram. The connection does not have any semantic or behavioral meaning, but just shows there is a relationship between the two objects. The **Association** is typically used with a Text Annotation to association text with an object (see table row below).	
Text Annotation (attached with an Association) (see the **SCE** specification)	**Text Annotations**, defined in **SCE**, are a mechanism for a modeler to provide additional information for the reader of a Shared Data Model.	
Group (see the **SCE** specification)	A **Group** object, defined in **SCE**, is a graphical box that surrounds the Shared Data Model behavioral elements. There are no specific semantics associated with **Groups**. However, a **Group** can be associated with a Participant of the Knowledge Package (such as the Processes of a Primary Care Provider and the Processes of a specialist).	

11.3 Connectors

11.3.1 Connector

The Connector element is the abstract class that provides the common properties for the four concrete types of connectors (listed in the next four sections). It is contained in a *SharedDataModel*.

The following figure shows the metamodel elements related to the *Connectors* element.

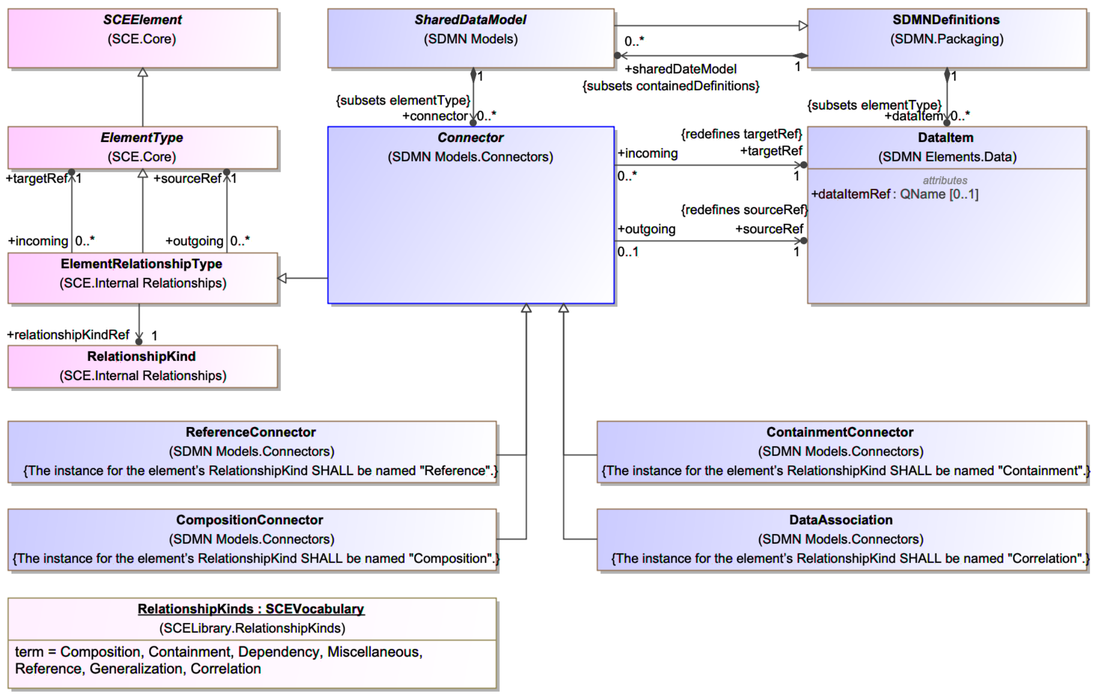


Figure 22: The Connectors Metamodel

Generalizations

The *Connector* element inherits the attributes and/or associations of:

- *ElementRelationshipType* (see the **SCE** Specification for more information).

Further, the *ElementRelationshipType* element inherits the attributes and/or associations of:

- *ElementType* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Properties

The following table presents the additional attributes and/or associations for *Connector*:

Table 21. Connector Attributes and/or Associations

Property/Association	Description
sourceRef : DataItem [1]	The **DataItem** that the *Connector* is connecting from.
targetRef : DataItem [1]	The **DataItem** that the *Connector* is connecting to.

11.3.2 CompositionConnector

A **CompositionConnector** is used to define a relationship between **DataItems**. This relationship will specify that one **DataItem** is contained within another **DataItem**. This relationship supports the composition of **DataItems**.

An example for **DataItem** composition can be found in healthcare scenarios: e.g., a patient record **DataItem** can be very complex and often only a portion of that **DataItem** is required for a **DMN** Decision. For example, creating a separate **DataItem** just for "demographics", which is part of (contained by) a larger "health record" **DataItem**, will help focus the model for the context that is being modeled at that point. This will help modelers and readers of the models to have a better understanding of the behaviors.

They are contained in a *SharedDataModel*.

The runtime consequences of creating a *RelationshipKind* of `Composition` through a **CompositionConnector** between two **DataItems** include:

- If a container is deleted, then the **DataItems** within the container will also be deleted.
- If a container is moved or set within another container, then the **DataItems** within the container will also be moved.
- If element within a folder-type **DataItem** container is updated (e.g., through a change in the value of a property), the container will not be updated. I.e., the container is not aware of changes to existing contained **DataItems**.
- If **DataItem** within a *non*-folder-type **DataItem** container is updated (e.g., through a change in the value of a property), the container will also be updated. I.e., the container is aware of changes to contained **DataItems**.

The following figure shows the metamodel elements related to the **CompositionConnector** element.

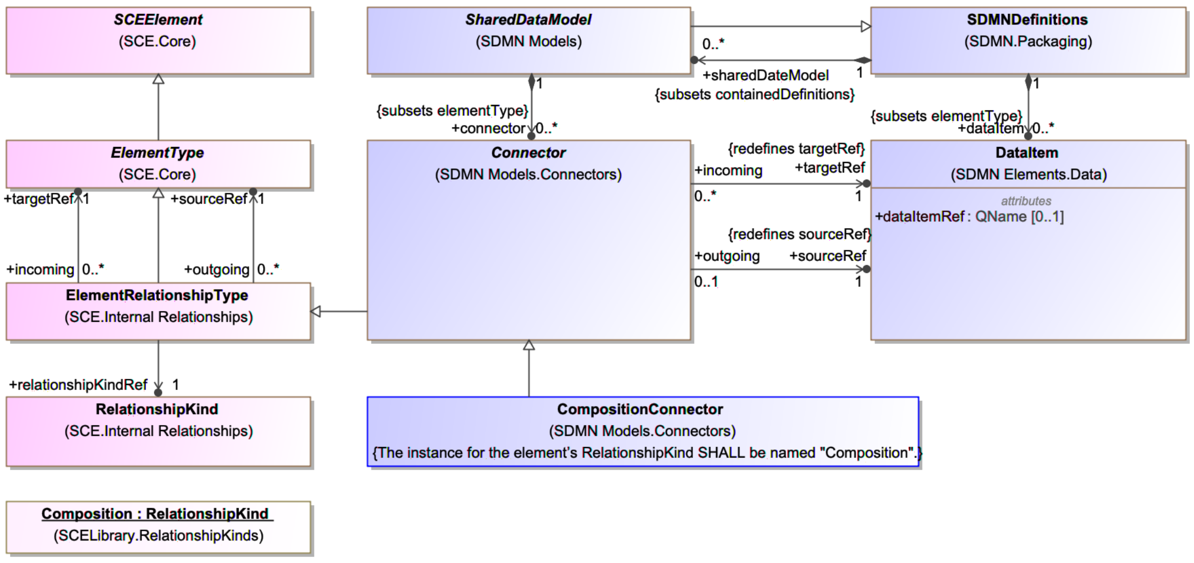


Figure 23: The CompositionConnector Metamodel

The **CompositionConnector** element does not have any additional attributes and/or associations.

Notation

The following statements define the notation for a **CompositionConnector**:

- A **CompositionConnector** is a line that SHALL be drawn with a single line (see below) with a filled diamond start and an angle 45° arrowhead end.

The use of text, color, size, and lines for a **CompositionConnector** SHALL follow the rules defined in the section entitled "

  - Use of Text, Color, Size, and Lines in a Diagram" above.

The following figure displays an example of a **CompositionConnector**:

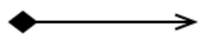


Figure 24: A Composition Connector

Connection Rules

The following statements define connection rules for a **CompositionConnector**:

- The source of a **CompositionConnector** SHALL be a **DataItem**.
- The target of a **CompositionConnector** SHALL be a **DataItem**.

Generalizations

The **CompositionConnector** element inherits the attributes and/or associations of:

- *Connector* (see the section entitled "Connector" for more information).

Further, the *Connector* element inherits the attributes and/or associations of:

- *ElementRelationshipType* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Constraints

A **CompositionConnector** is a type of **SCE** *ElementRelationshipType*, but is distinguished with this constraint:

- The instance for the element's *RelationshipKind* SHALL be named "Composition".

Properties

The **Composition Connector** element does not have any additional attributes and/or associations.

11.3.3 ContainmentConnector

A **ContainmentConnector** is used to define a relationship between **DataItems**. This relationship will specify that one **DataItem** is contained within another **DataItem**. This relationship supports the containment of **DataItems**, including the parent and child association that exists between **CMMN** CaseFileItems.

They are contained in a *SharedDataModel*.

The runtime consequences of creating a *RelationshipKind* of `Containment` through a **ContainmentConnector** between two **DataItems** include:

- If a container is deleted, then the **DataItems** within the container will also be deleted.
- If a container is moved or set within another container, then the **DataItems** within the container will also be moved.
- If element within a folder-type **DataItem** container is updated (e.g., through a change in the value of a property), the container will not be updated. I.e., the container is not aware of changes to existing contained **DataItems**.
- If **DataItem** within a *non*-folder-type **DataItem** container is updated (e.g., through a change in the value of a property), the container will also be updated. I.e., the container is aware of changes to contained **DataItems**.

The following figure shows the metamodel elements related to the **ContainmentConnector** element.

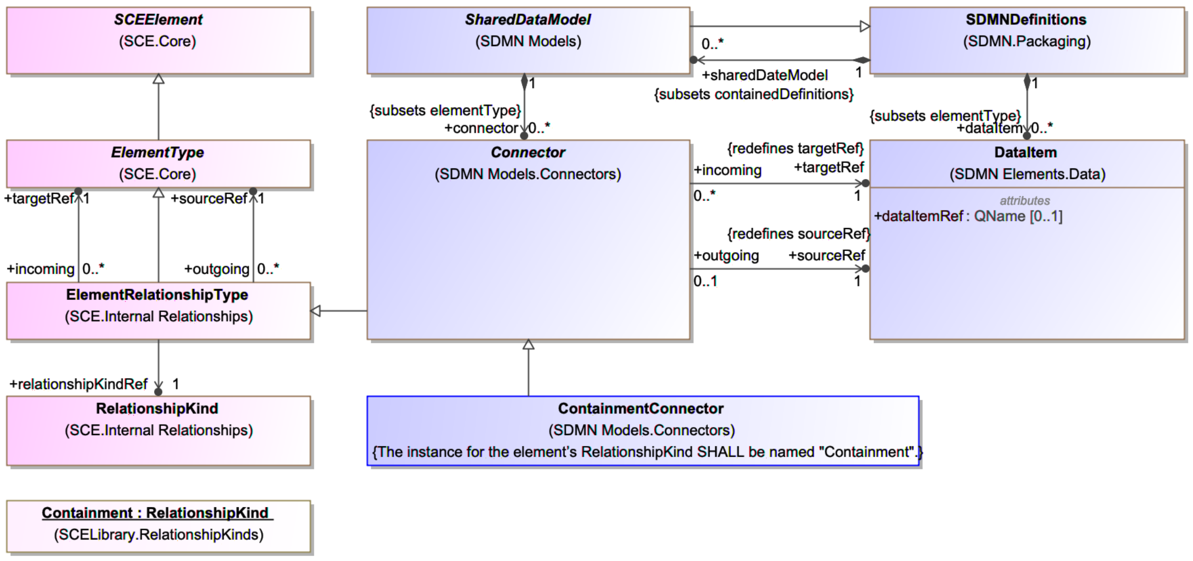


Figure 25: The Containment Connector Metamodel

Notation

The following statements define the notation for a **ContainmentConnector**:

- A **ContainmentConnector** is a line that SHALL be drawn with a single line (see below) with a cross-filled circle start and without an arrowhead end.

The use of text, color, size, and lines for a **ContainmentConnector** SHALL follow the rules defined in the section entitled "

  - o Use of Text, Color, Size, and Lines in a Diagram" above.

The following figure displays an example of a **ContainmentConnector**:

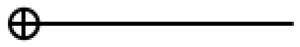


Figure 26: A Containment Connector

Connection Rules

The following statements define connection rules for a **ContainmentConnector**:

- The source of a **ContainmentConnector** SHALL be a **DataItem**.
- The target of a **ContainmentConnector** SHALL be a **DataItem**.

Generalizations

The **ContainmentConnector** element inherits the attributes and/or associations of:

- *Connector* (see the section entitled "Connector" for more information).

Further, the *Connector* element inherits the attributes and/or associations of:

- *ElementRelationshipType* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Constraints

A **ContainmentConnector** is a type of **SCE** *ElementRelationshipType*, but is distinguished with this constraint:

- The instance for the element's *RelationshipKind* SHALL be named "Containment".

Properties

The **ContainmentConnector** element does not have any additional attributes and/or associations.

11.3.4 DataAssociation

The **DataAssociation** class is a *Connector* and used to model how data is mapped between two **DataItems**. The source of the association is mapped to the target. The *ItemDefinition* from the souceRef and targetRef MUST have the same ItemDefinition or the **DataAssociation** MUST have a transformation *Expression* that transforms the source ItemDefinition into the target ItemDefinition. It is contained within a *SharedDataModel*.

The following figure shows the metamodel elements related to the **DataAssociation** element.

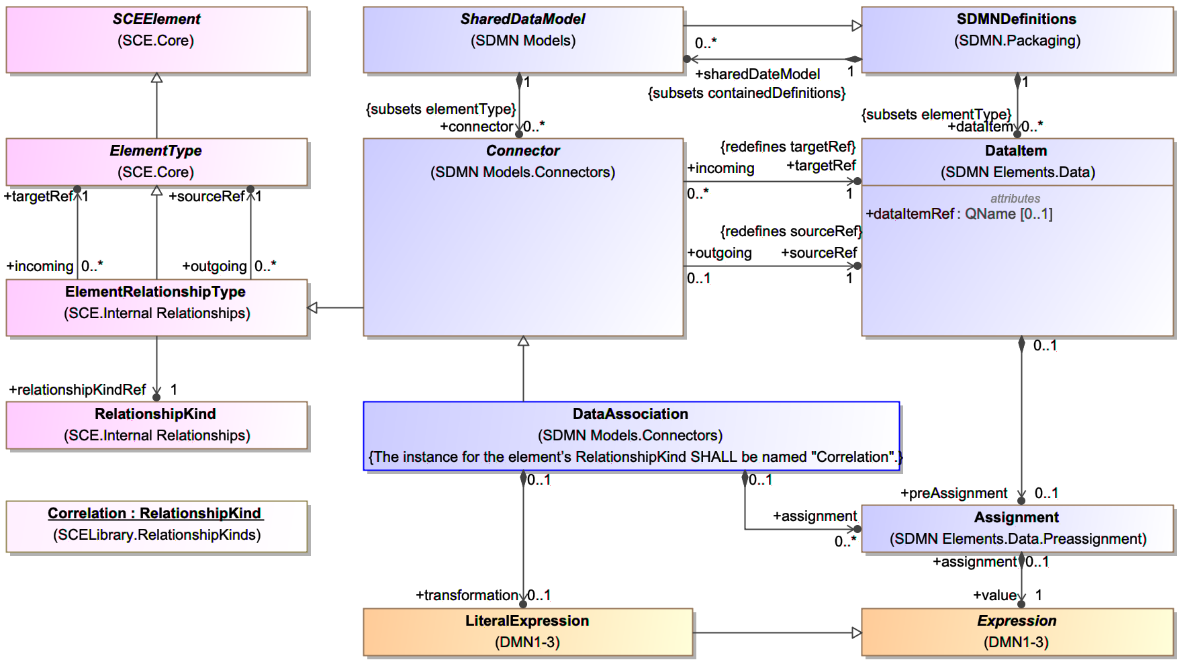


Figure 27: The DataAssociation Metamodel

Notation

The following statements define the notation for a **DataAssociation**:

- A **DataAssociation** is a line that SHALL be drawn with a dotted single line (see below) with an angle 45° arrowhead end.
    - Note that the line style of the Data Association is the same as the **SCE** Model Artifact, **Association**. This graphical overlap was included in the **BPMN 2.0** specification and **SDMN** was designed to be consistent with other **BPM+** specifications. Thus, the same graphical overlap is being applied.
        - If a line that looks like an **Association** or a **DataAssociation** is connected between two **DataItems**, then the connector is assumed to be a **DataAssociation** (see the section entitled "Data Association Connection Rules," below). If the source or target of the line is not a **DataItem**, then the connector is assumed to be an **Association**.

The use of text, color, size, and lines for a **DataAssociation** SHALL follow the rules defined in the section entitled "

- o Use of Text, Color, Size, and Lines in a Diagram" above.
- The arrowhead of the connector is attached to the **DataItem** that is the target of the data mapping.

The following figure displays an example of a **DataAssociation**:

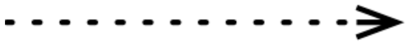


Figure 28: A Data Association

Connection Rules

The following statements define connection rules for a **DataAssociation** connector:

- The source of a **DataAssociation** SHALL be a **DataItem**.
- The target of a **DataAssociation** SHALL be a **DataItem**.

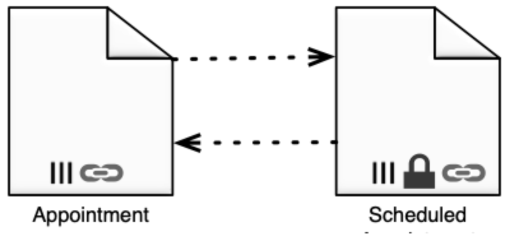


Figure 29: Example of DataAssociations between two DataItems

Generalizations

The **DataAssociation** element inherits the attributes and/or associations of:

- *Connector* (see the section entitled "Connector" for more information).

Further, the *Connector* element inherits the attributes and/or associations of:

- *ElementRelationshipType* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Constraints

A **DataAssociation** is a type of **SCE** *ElementRelationshipType*, but is distinguished with this constraint:

- The instance for the element's *RelationshipKind* SHALL be named "Correlation".

Properties

The following table presents the additional attributes and/or associations for **DataAssociation**:

Table 22. DataAssociation Attributes and/or Associations

Property/Association	Description
assignment : Assignment [0..*]	Specifies one or more data elements *Assignments*. By using an *Assignment*, single data structure elements can be assigned from the source structure to the target structure.
transformation : LiteralExpression [0..1]	Specifies an optional transformation *Expression*. The actual scope of accessible data for that *Expression* is defined by the source and target of the specific **DataAssociation** types.

11.3.5 ReferenceConnector

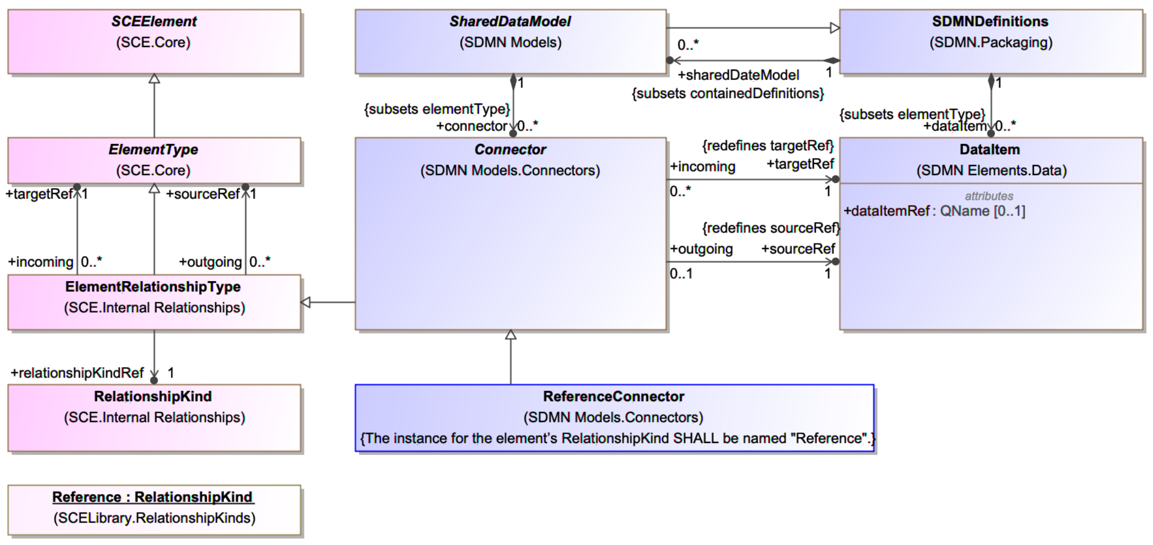
An *ReferenceRelationship* is used to define a relationship between **DataItems**. This relationship will specify that one **DataItem** is connected in some way to another **DataItem** – there is some type of relationship. This relationship is included to support the source and target reference associations that exists between **CMMN** CaseFileItems. For example, the CaseFileItems might be created at the same time (although independently) or they are often sent together during the performance of a **CMMN** Case, etc. In a sense, it is a non-graphical way of grouping **DataItems**.

They are contained in a *SharedDataModel*.

The runtime consequences of creating a *RelationshipKind* of `Reference` through a **CompositionConnector** between two **DataItems** include:

- If a **DataItem** that is referenced by another **DataItem** (either as a source or target) is deleted, then the referenced **DataItems** will not be deleted.
- If a **DataItem** is moved or set within another container, then the **DataItems** referenced by that **DataItem** will not be moved.
- If **DataItem** is updated (e.g., through a change in the value of a property), the **DataItems** referenced by that **DataItem** will not be updated. I.e., a **DataItem** is not aware of changes to any referenced **DataItems**.

The following figure shows the metamodel elements related to the **ReferenceConnector** element.



SCEElement (SCE.Core)
SharedDataModel (SDMN Models)
SDMNDefinitions (SDMN.Packaging)
0..*
+sharedDateModel 1
{subsets containedDefinitions}
{subsets elementType} +connector 0..*
{subsets elementType} +dataItem 0..*
ElementType (SCE.Core)
Connector (SDMN Models.Connectors)
DataItem (SDMN Elements.Data)
attributes
+dataItemRef : QName [0..1]
{redefines targetRef} +incoming +targetRef 0..* 1
{redefines sourceRef} +outgoing +sourceRef 0..1 1
+targetRef 1
+sourceRef 1
+incoming 0..*
+outgoing 0..*
ElementRelationshipType (SCE.Internal Relationships)
+relationshipKindRef 1
RelationshipKind (SCE.Internal Relationships)
ReferenceConnector (SDMN Models.Connectors)
{The instance for the element's RelationshipKind SHALL be named "Reference".}
Reference : RelationshipKind (SCELibrary.RelationshipKinds)

Figure 30: The Reference Connector Metamodel

Notation

The following statements define the notation for a **ReferencConnector**:

- A **ReferenceConnector** is a line that SHALL be drawn with a long dashed single line (see below) with an angle 45° arrowhead end.

The use of text, color, size, and lines for a **ReferenceConnector** SHALL follow the rules defined in the section entitled "

  - Use of Text, Color, Size, and Lines in a Diagram" above.

The following figure displays an example of a **ReferenceConnector**:

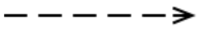


Figure 31: A Reference Connector

Connection Rules

The following statements define connection rules for a **ReferenceConnector**:

- The source of a **ReferenceConnector** SHALL be a **DataItem**.
- The target of a **ReferenceConnector** SHALL be a **DataItem**.

Generalizations

The **ReferenceConnector** element inherits the attributes and/or associations of:

- *Connector* (see the section entitled "Connector" for more information).

Further, the *Connector* element inherits the attributes and/or associations of:

- *ElementRelationshipType* (see the **SCE** Specification for more information [OMG doc number bmi-2021-12-09]).

Constraints

A **ReferenceConnector** is a type of **SCE** *ElementRelationshipType*, but is distinguished with this constraint:

- The instance for the element's *RelationshipKind* SHALL be named "Reference".

Properties

The **ReferenceConnector** element does not have any additional attributes and/or associations.

11.4 Model Artifacts

SDMN provides modelers with the capability of showing additional information about a **Shared Data Model** that is not directly related to the model elements through the capability provided by the *ModelArtifact* elements that are defined in the **SCE** specification. **SDMN** utilizes the three standard **SCE** *ModelArtifacts*: **Associations**, **Groups**, and **TextAnnotations**.

SDMN does not extend the capabilities of these *ModelArtifacts* but uses them as-is from the **SCE** specification. The following figure shows how the **SCE** *ModelArtifact* is included within **SDMN**. *ModelArtifacts* are contained within a **SDMNModel**.

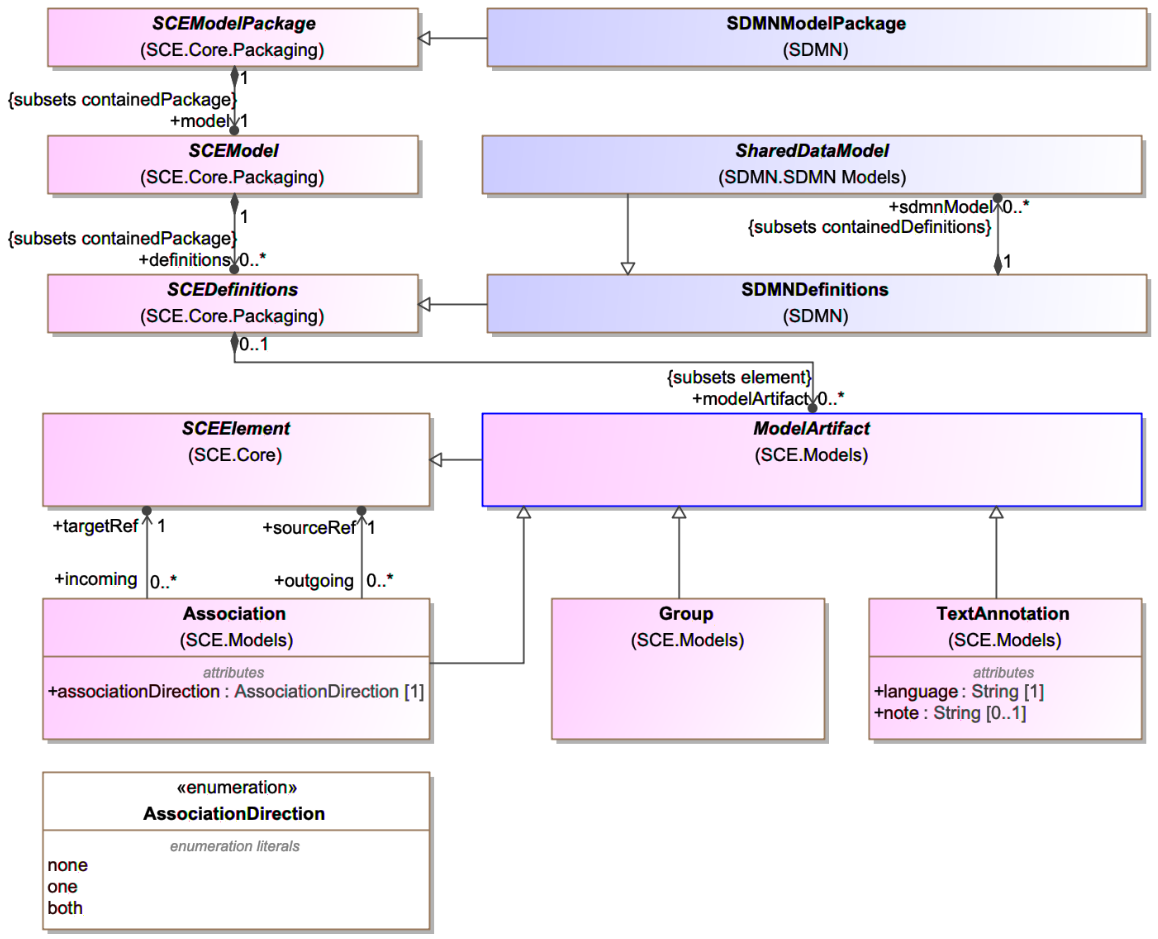


Figure 32: The Use of SCE Artifacts in SDMN

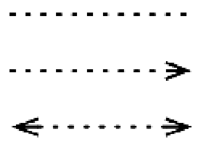
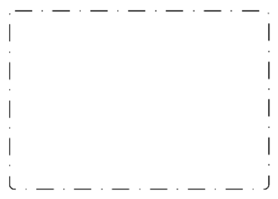
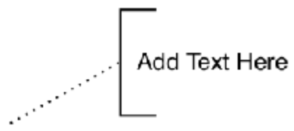
A modeler or modeling tool MAY extend a **SDMN** Model and add new types of *ModelArtifacts*. Any new *ModelArtifacts* SHALL follow the *Connector* connection rules (listed below). **Associations** can be used to link *ModelArtifacts* to other Model elements.

Notation

Full details of Model Artifacts are available in the section entitled "ElementType", above, but the notation of the elements is provided here for convenience.

The table below displays the graphical elements of **SCE**'s Model Artifacts:

Table 23. Shared Data Model Graphical Elements

Element	Description	Notation
Association	An **Association** is used to associate Model Artifacts (often **Text Annotations**) or model elements to other model elements. The connection only specifies that there is some relationship between the two elements, but no model semantics are implied. An **Association** is line that is drawn with a dotted single line. An angle 30° arrowhead may optionally be added to either end of the line.	
Group	The **Group** object is a Model Artifact that provides a visual mechanism to group elements of a Model informally. **Groups** are often used to highlight certain sections of a Model without adding additional semantics. The highlighted (grouped) section of the Model can be separated for reporting and analysis purposes. A **Group** is a rounded corner rectangle that is drawn with a solid dashed and dotted line (see figure to the right).	
Text Annotation	**Text Annotations** are a mechanism for a modeler to provide additional information for the reader of a **SDMN** Model. An **Association** may be used to connect user-defined text (a **Text Annotation**) with a Model element. A **Text Annotation** is an open rectangle that is drawn with a solid single line (see figure to the right).	

Model Artifact Connection Rules

The following statements define connection rules for a *ModelArtifact*:

- A *ModelArtifact* SHALL NOT be a target for a **CompositionConnector**, a **ContainmentConnector**, a **DataAssociation**, or a **ReferenceConnector** .
- A *ModelArtifact* SHALL NOT be a source for a **CompositionConnector**, a **ContainmentConnector**, a **DataAssociation**, or a **ReferenceConnector**.

12 SDMN Library

A Library is included in **SDMN** to provide standard instances that should be implemented by tools supporting **SDMN**. Currently, **SDMN** defines the instances for two sub-packages named *ItemKinds* and *MultiplicyKinds* (See next two sections).

12.1 ItemKinds

The following figure presents the instances for the *ItemKind* element that are `terms` for the instance (*ItemKinds*) of the *SDMNVocabulary* element:

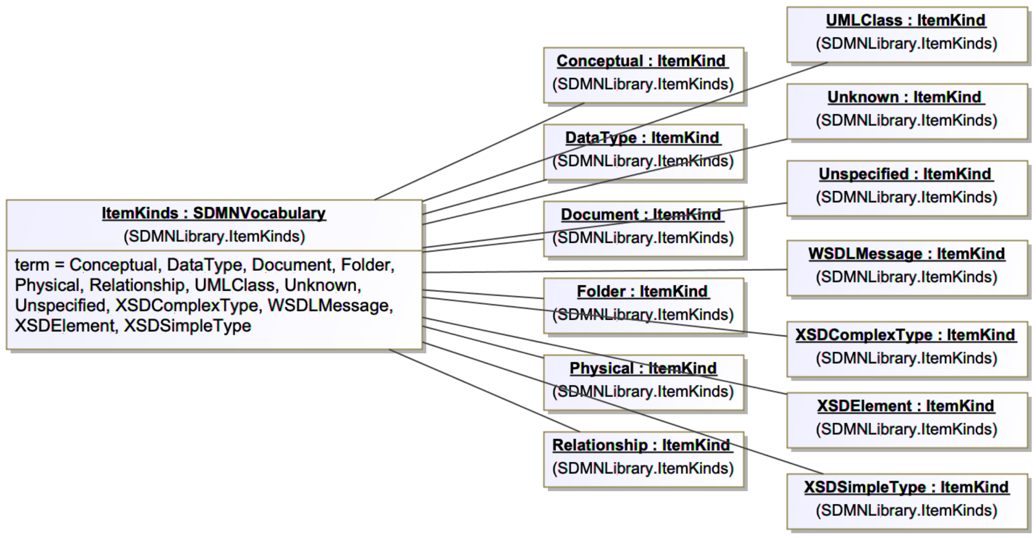


Figure 33: The ItemKinds Instance Model

Some of the literals for *ItemKind* are based on the **CMMN** CaseFileItemDefinition literals for `DefinitionType`.

The following table presents a description for the included instances for *ItemKind*:

Table 24. ItemKind Literals

Literal	Description
Conceptual	The type of the *ItemDefinition* that doesn't represent data or physical items, but represents concepts in the minds of users that are important for tasks or decisions. For example, a preference for a particular type of procudure will influence a doctor's decision. While actual computations cannot be made with `Conceptual` *ItemDefinitions*, they are used to document aspects of the modeled behaviors.
DataType	The type of the ItemDefinition that fully utilizes the structural data capabilities inherent to ItemDefinition. Using FEEL, these will define simple types or data structures.
Document	This represents a Data Object or Case File Item that is a type of `Document`. In **BPMN**, the document could be physical (e.g., printed) or electronic. In **CMMN**, it would represent a document in a Document Management System and is defined through the following URI: http://www.omg.org/spec/CMMN/DefinitionType/CMISDocument
Folder	This represents a **CMMN** Case File Item that is a `Folder`. A `Folder` can contain other `Folders` or `Documents`. These relationships are set through the `Child` and `Parent` attributes of the **DataItem**. Neither **BPMN** nor **DMN** have the concept of `Folder` as a data element. Thus, **DataItems** based on a `Folder` *ItemDefinition* would not map to **BPMN** or **DMN** data elements. The `Folder` is defined through the following URI: http://www.omg.org/spec/CMMN/DefinitionType/CMISFolder

Physical	The *ItemKind* is represents objects in a **BPMN** Process that are physical objects, such as printed documents or manufactured items. These types of **DataItems** are not currently relevant to **CMMN** or **DMN**.
Relationship	The *ItemKind* is defined through the following URI: http://www.omg.org/spec/CMMN/DefinitionType/CMISRelationship
UMLClass	The *ItemKind* is represents a **UML** Class in a Class Diagram.
Unknown	The *ItemKind* is defined through the following URI: http://www.omg.org/spec/CMMN/DefinitionType/Unknown
Unspecified	The *ItemKind* is defined through the following URI: http://www.omg.org/spec/CMMN/DefinitionType/Unspecified
WSDLMessage	The *ItemKind* is represents a **WSDL** Message.
XSDComplexType	For *ItemKinds* of this type, the (**SCE**) *Import* class SHOULD be used to import an XML Schema definition into the **Shared Data Model**. The *ItemKind* is defined through the following URI: http://www.omg.org/spec/CMMN/DefinitionType/XSDComplexType
XSDElement	For *ItemKinds* of this type, the (**SCE**) *Import* class SHOULD be used to import an XML Schema definition into the **Shared Data Model**. The *ItemKind* is defined through the following URI: http://www.omg.org/spec/CMMN/DefinitionType/XSDElement
XSDSimpleType	For *ItemKinds* of this type, the (**SCE**) *Import* class SHOULD be used to import an XML Schema definition into the **Shared Data Model**. The *ItemKind* is defined through the following URI: http://www.omg.org/spec/CMMN/DefinitionType/XSDSimpleType

12.2 Multiplicity Kinds

The following figure presents the instances for the *MultiplicyKind* element that are `terms` for the instance (*MultiplicyKinds*) of the *SDMNVocabulary* element:

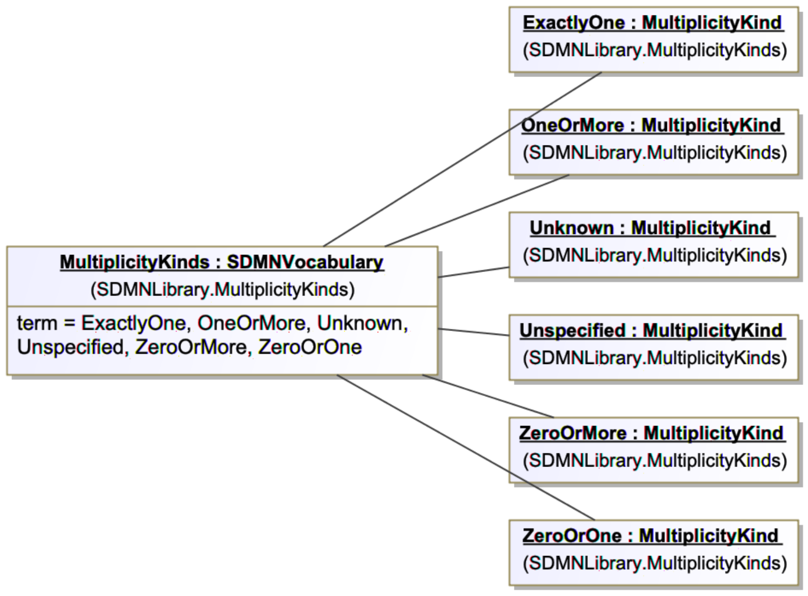

ExactlyOne : MultiplicityKind
(SDMNLibrary.MultiplicityKinds)
OneOrMore : MultiplicityKind
(SDMNLibrary.MultiplicityKinds)
Unknown : MultiplicityKind
(SDMNLibrary.MultiplicityKinds)
MultiplicityKinds : SDMNVocabulary
(SDMNLibrary.MultiplicityKinds)
term = ExactlyOne, OneOrMore, Unknown, Unspecified, ZeroOrMore, ZeroOrOne
Unspecified : MultiplicityKind
(SDMNLibrary.MultiplicityKinds)
ZeroOrMore : MultiplicityKind
(SDMNLibrary.MultiplicityKinds)
ZeroOrOne : MultiplicityKind
(SDMNLibrary.MultiplicityKinds)

Figure 34: The MultiplcityKinds Instance Model

The following table presents a description for the included instances for *MultiplicyKind*:

Table 25. MultiplicityKind Literals

Literal	Description
ExactlyOne	There is one copy of this *ItemDefintion* or **DataItem**.
OneOrMore	There is at least one copy of this *ItemDefintion* or **DataItem**, but there may be more.
Unknown	The muliplicty is not know for this *ItemDefintion* or **DataItem**.
Unspecified	The muliplicty is not specified for this *ItemDefintion* or **DataItem**.
ZeroOrMore	There may be no copies of this *ItemDefintion* or **DataItem** or there may be multiple copies.
ZeroOrOne	There may be no copies of this *ItemDefintion* or **DataItem** or there may be one copy.

13 Mapping to BPM+ Models

The elements of **SDMN**, especially **DataItems** and *ItemDefintions*, are intended for use by **BPMN**, **CMMN**, and **DMN** models. There are differences between the way data is used and defined across these three types of models. Thus, if **SDMN** is going to support all of them, then the **SDMN** must in fact define data elements as a super-set of the capabilities of the other three models.

The following sub-sections define any mappings required for **SDMN** data elements to be imported into the other BPM+ models.

13.1 Element Terminology Mapping to BPM+ Element Terminology

The following table defines the mapping for terms across the BPM+ languages and **SDMN**:

Table 26. Mapping to BPMN

SDMN Element	BPMN Element	CMMN Element	DMN Element
Containment Connector	N/A	N/A	N/A
Composition Connector	N/A	N/A	N/A
Data Association	Data Association	N/A	N/A
DataItem	Item Aware Element) Data Object, Data Object Reference, Data Input, Data Output, Data Store, Data Store Reference, Property)	Case File Item	Information Item (Data Input, Decision Output)
Data State	Data State (for Item Aware Element)	N/A	N/A
ItemDefinition	ItemDefinition	CaseFileItemDefinition	ItemDefinition
ItemKind	ItemKind	definitionType	N/A
Multiplicity	collection	multiplicity	N/A
Pre-Assignment	N/A	N/A	N/A
Reference Connector	N/A	N/A	N/A
SDMNModelPackage	Definitions	Definitions	Definitions

13.2 BPMN

This section provides mapping from **SDMN** to **BPMN**.

ItemKind Mapping

The following table defines the mapping from **SDMN** *ItemKind* instances to **BPMN** itemKind literals:

Table 27. ItemKind Mapping

SDMN ItemKinds	BPMN itemKind Literals
Conceptual	Information (this is to allow a formal documentation of the characteristics of the `Conceptual` **DataItem**.)
DataType	Information
Document	Information
Folder	Information
Physical	Physical
Relationship	Information

UMLClass	Information
Unknown	Information
Unspecified	Information
WSDLMessage	Information
XSDComplexType	Information
XSDElement	Information
XSDSimpleType	Information

MultiplicityKind Mapping

The `multiplicity` attribute for the **SDMN** *ItemDefinition* element and the **DataItem** element is consistent with the **CMMN** `multiplicity` attribute for a Case File Item. However, the attribute is not consistent with the `isCollection` attribute for the *ItemDefinition* element of both **BPMN**. But the `multiplicity` values can be mapped to the Boolean `isCollection`.

The following table defines the mapping from **SDMN** *MultiplicityKind* instances to the **BPMN** Collection property:

Table 28. MultiplicityKind Mapping

SDMN MultiplicityKinds	BPMN Collection Boolean
ZeroOrOne	False An actual value of zero would not be valid for **BPMN** data elements. Thus, it is recommended to avoid this setting for **SDMN DataItems** that are used in **BPMN** models.
ZeroOrMore	False An actual value of zero would not be valid for **BPMN** data elements. Thus, it is recommended to avoid this setting for **SDMN DataItems** that are used in **BPMN** models.
ExactlyOne	False
OneOrMore	True
Unspecified	False This setting implies that the actual value could be zero, which would not be valid for **BPMN** data elements. Thus, it is recommended to avoide this setting for **SDMN DataItems** that are used in **BPMN** models.
Unknown	False This setting implies that the actual value could be zero, which would not be valid for **BPMN** data elements. Thus, it is recommended to avoide this setting for **SDMN DataItems** that are used in **BPMN** models.

Element Mapping

The following table defines the mapping from **SDMN** elements and attributes to **BPMN**:

Table 29. Mapping to BPMN

SDMN Element/Attribute	BPMN Element/Attribute
DataItem (not ItemKind Folder)	Data Object
DataItem (ItemKind Folder)	N/A
Item Definition	Item Definition If the *ItemDefinition* for a **DataItem** is of type `definitionRef`, then the *ItemDefinition* will mapped to a **BPMN** *ItemDefinition* for an Data Object. If the *ItemDefinition* for a **DataItem** is of type `metaDefinitionRef`, then the contents of the *ItemDefinition* will be ignored. There is no equivalent in **BPMN**. A separate *ItemDefinition* will not be created.
DataItem/Location	N/A
DataItem/ItemFormat	N/A
DataState	DataState
DataItem/preAssignment	N/A Although imlementations of **BPMN** could establish an activity at the beginning of the process that fills the output Data Object with the values listed in the pre-assignment.

13.3 CMMN

This section provides mapping from **SDMN** to **CMMN**.

ItemKind Mapping

The following table defines the mapping from **SDMN** *MultiplicityKind* instances to the **BPMN** Collection property:

Table 30. ItemKind Literals

SDMN itemKind Literals	CMMN itemKind Literals
Conceptual	Unknown
DataType	Unspecified
Document	Document in CMIS
Folder	Folder in CMIS
Physical	Unspecified
Relationship	Relationship in CMIS
UMLClass	Unspecified
Unknown	Unknown
Unspecified	Unspecified

WSDLMessage	WSDLMessage
XSDComplexType	XML Schema Complex Type
XSDElement	XML-Schema Element
XSDSimpleType	XML Schema Simple Type

Element Mapping

The following table defines the mapping from **SDMN** elements and attributes to **CMMN**:

Table 31. Mapping to CMMN

SDMN Element/Attribute	CMMN Element/Attribute
DataItem (not ItemKind Folder)	CaseFileItem (not ItemKind Folder)
DataItem (ItemKind Folder)	CaseFileItem (ItemKind Folder)
Item Definition	CaseFileItemDefinition If the *ItemDefinition* for a **DataItem** is of type `definitionRef`, then the *ItemDefinition* will mapped to a **CMMN** *ItemDefinition* for an CaseFileItem. If the *ItemDefinition* for a **DataItem** is of type `metaDefinitionRef`, then the simple types of the *ItemDefinition* will mapped to **CMMN** properties for an CaseFileItem.
DataItem/Location	N/A
DataItem/ItemFormat	N/A
DataState	N/A But perhaps could be reflected through a Milestone.
DataItem/preAssignment	N/A

13.4 DMN

This section provides mapping from **SDMN** to **DMN**.

MultiplicityKind Mapping

The `multiplicity` attribute for the **SDMN** *ItemDefinition* element and the **DataItem** element is consistent with the **CMMN** `multiplicity` attribute for a Case File Item. However, the attribute is not consistent with the `isCollection` attribute for the *ItemDefinition* element of both **BPMN** and **DMN**. But the `multiplicity` values can be mapped to the Boolean `isCollection`.

The following table defines the mapping from **SDMN** *MultiplicityKind* instances to the **DMN** Collection property:

Table 32. MultiplicityKind Mapping

SDMN MultiplicityKinds	DMN Collection Boolean
ZeroOrOne	False An actual value of zero would not be valid for **DMN** data elements. Thus, it is recommended to avoid this setting for **SDMN DataItems** that are used in **DMN** models.
ZeroOrMore	False An actual value of zero would not be valid for **DMN** data elements. Thus, it is recommended to avoid this setting for **SDMN DataItems** that are used in **DMN** models.
ExactlyOne	False
OneOrMore	True
Unspecified	False This setting implies that the actual value could be zero, which would not be valid for **DMN** data elements. Thus, it is recommended to avoide this setting for **SDMN DataItems** that are used in **DMN** models.
Unknown	False This setting implies that the actual value could be zero, which would not be valid for **DMN** data elements. Thus, it is recommended to avoide this setting for **SDMN DataItems** that are used in **DMN** models.

Element Mapping

The following table defines the mapping from **SDMN** elements and attributes to **DMN**:

Table 33. Mapping to DMN

SDMN Element/Attribute	DMN Element/Attribute
DataItem (not ItemKind Folder)	InformationItem
DataItem (ItemKind Folder)	N/A
Item Definition	Item Definition If the *ItemDefinition* for a InformationItem is of type `definitionRef`, then the *ItemDefinition* will mapped to a **DMN** *ItemDefinition* for an InformationItem. If the *ItemDefinition* for a **DataItem** is of type `metaDefinitionRef`, then the contents of the *ItemDefinition* will be ignored. There is no equivalent in **DMN**. A separate *ItemDefinition* will not be created.
DataItem/ItemKind	N/A
DataItem/Location	N/A
DataItem/ItemFormat	N/A
DataState	N/A

DataItem/preAssignment	N/A

14 SDMN Examples

14.1 Hello Patient

The following figure provides an example of how a **SDMN** Data Item Diagram could look. It is based on a sample use case named “Hello Patient”, which is BPM+ Knowledge Package definition of a visit to a doctor’s office (which is BPM+ modeling technique for representing clinical guidelines). The **DataItems** in the model support the data elements of the Process, Case, and Decision models that define the behavioral components of the Knowledge Package.

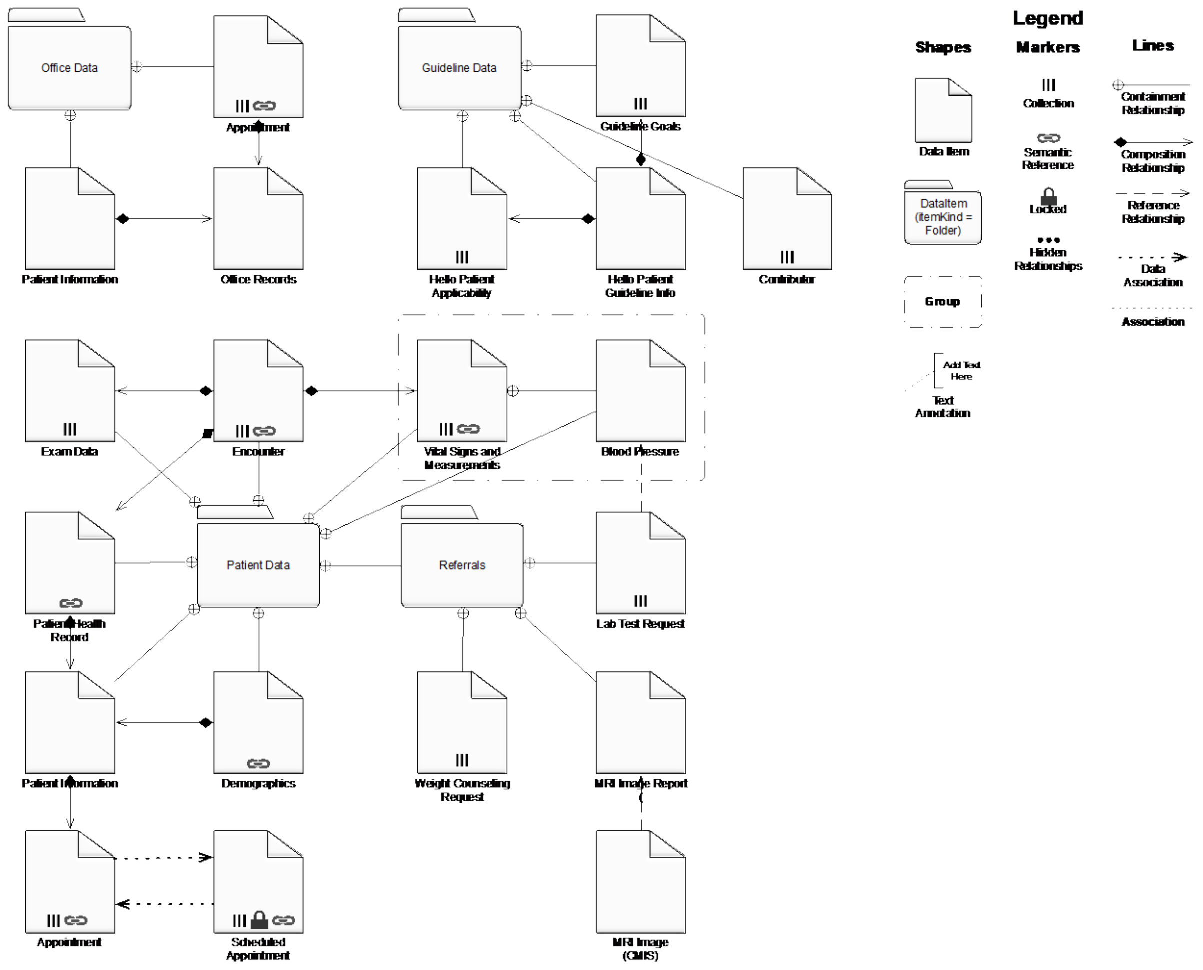


Figure 35: An Example of How a SDMN DataItem Diagram

The following table lists the data elements used by the BPM+ models of the Knowledge Package.

Table 34. List of Data Elements used by the BPM+ Models in the Hello Patient Use Case

Cases	Decision Services	Processes
1. Appointment	**1. Blood Pressure**	**1. Appointment**
2. Blood Pressure	2. Blood Pressure Goal	**2. Blood Pressure**
3. Blood Pressure Goal	3. Blood Pressure Rating	3. Blood Pressure Goal
4. BMI Category	4. BMI Category	4. Blood Pressure Rating
5. Contributor	**5. Demographics**	5. BMI Category
6. Encounter	**6. Exam Data**	**6. Contributor**
7. Exam Data	7. Health Conditions	**7. Demographics**
8. Guideline Goals	8. Medication	**8. Encounter**
9. Health Conditions	9. Medication Tolerances	**9. Exam Data**
10. Hello Patient Applicable	10. Pathway Goals	10. Health Conditions
11. Lab Test Request	11. Patient Complaints	**11. Hello Patient Applicable**
12. Medication	**12. Patient Health Record**	**12. Hello Patient Guideline Goals**
13. Medication Tolerances	13. Referral	**13. Lab Test Request**
14. MRI Image	14. Treatment Plan	14. Loop Counter
15. MRI Image Report	15. Treatment Choice	15. Medication
16. Office Data	**16. Vital Signs and Measurements**	16. Medication Tolerances
17. Office Records	**17. Weight Counseling Request**	**17. MRI Image**
18. Pathway Goals	18. Weight Counseling Request Choice	**18. MRI Image Report**
19. Patient Data		**19. Office Records**
20. Patient Health Record		20. Pathway Goals
21. Patient Information		21. Patient Complaints
22. Referrals		**22. Patient Health Record**
23. Scheduled Appointment		**23. Patient Information**
24. Treatment Plan		24. Referral
25. Vital Signs and Measurements		**25. Scheduled Appointment**
26. Weight Counseling Request		26. Treatment Plan
		27. Treatment Choice
		28. Vital Signs and Measurements
		29. Weight Counseling Request
		30. Weight Counseling Request Choice

Note that the data elements listed in **bold** in the table are those that appear in the **SDMN** model.

15 Exchange Formats

15.1 Interchanging Incomplete Models

In practice, it is common for 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 person) first defines a high-level model, and then passes it on to another user to be completed and refined.

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

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

- Disregarding missing attributes that are marked as 'required' in the XSD.
- Reducing the lower bound of elements with 'minOccurs' greater than 0.

15.2 XSD

15.2.1 Document Structure

A domain-specific set of model elements is interchanged in one or more **SDMN** files. The root element of each file SHALL be `<DMN:Definitions>`. The set of files SHALL be self-contained, i.e., all definitions that are used in a file SHALL be imported directly or indirectly using the `<DMN:Import>` element.

Each file SHALL declare a "`namespace`" that MAY differ between multiple files of one model.

SDMN files MAY import non-**SDMN** files (such as XSDs and PMMLs) if the contained elements use external definitions.

15.2.2 References within the SDMN XSD

Many **SDMN** elements that may need to be referenced contain IDs and within the **SDMN** XSD, references to elements are expressed via these IDs. The XSD IDREF type is the traditional mechanism for referencing by IDs, however it can only reference an element within the same file. **SDMN** elements that inherit from **SCE** *RootElement* referencing by ID, across files, by utilizing an href attribute whose value must be a valid URI reference [RFC 3986] where the path components may be absolute or relative, the reference has no query component, and the fragment consists of the value of the id of the referenced **SDMN** element.

For example, consider the following Decision:

```
<decision name="Pre-Bureau Risk Category" id="prebureauriskDec01">…</decision>
```

When this Decision is referenced, e.g. by an InformationRequirement in a Decision that is defined in another file, the reference could take the following form:

```
<requiredDecision href="http://www.example.org/Definitions01.xml#prebureauriskDec01"/>
```

where "`http://www.example.org/Definitions01.xml`" is an URI reference to the XML document in which the "Pre-Bureau Risk Category" Decision is defined (e.g. the value of the locationURI attribute in the corresponding Import element), and "prebureauriskDec01" is the value of the id attribute for the Decision.

If the path component in the URI reference is relative, the base URI against which the relative reference is applied is determined as specified in [RFC 3986]. According to that specification, "if no base URI is embedded and the representation is not encapsulated within some other entity, then, if a URI was used to retrieve the representation, that URI shall be considered the base URI" ([RFC 3986], section 5.1.3). That is, if the reference is not in the scope of an xml:base attribute [XBASE], a value of the href attribute that contains only a fragment and no path component references a **SDMN** element that is defined in the same instance of XML file as the referencing element. In the example below, assuming that the requiredDecision element is not in the scope of an xml:base attribute, the **SDMN** element whose id is "prebureauriskDec01" must be defined in the same XML document:

```
<requiredDecision href="#prebureauriskDec01" />
```

Attribute typeRef references ItemDefinitions and built-in types by name not ID. In order to support imported types, typeRef uses the namespace-qualified name syntax [qualifer].[local-name], where qualifier is specified by the name attribute of the Import element for the imported type. If the referenced type is not imported, the prefix SHALL be omitted.

16 SDMN Diagram Interchange (SDMN DI)

16.1 Scope

This chapter specifies the meta-model and schema for **SDMN** Diagram Interchange (**SDMN DI**). The **SDMN DI** is meant to facilitate the interchange of **SDMN** diagrams between tools rather than being used for internal diagram representation by the tools. The simplest interchange approach to ensure the unambiguous rendering of a **SDMN** diagram was chosen for **SDMN DI**. As such, **SDMN DI** does not aim to preserve or interchange any "tool smarts" between the source and target tools (e.g., layout smarts, efficient styling, etc.).

SDMN DI does not ascertain that the **SDMN** diagram is syntactically or semantically correct.

16.2 Diagram Definition and Interchange

The **SDMN DI** meta-model, similar to the **SDMN** abstract syntax meta-model, is defined as a MOF-based meta-model. As such, its instances can be serialized and interchanged using XMI. **SDMN DI** is also defined by an XML schema. Thus its instances can also be serialized and interchanged using XML.

Both **SDMN DI** meta-model and schema is layered upon the **SCE DI** (see the **SCE 1.0** specification), which is harmonized with the OMG Diagram Definition (DD) standard version 1.1. The referenced DD contains two main parts: the Diagram Commons (DC) and the Diagram Interchange (DI). The DC defines common types like bounds and points, while the DI provides a framework for defining domain-specific diagram models. As a domain-specific DI, **SDMN DI** defines a few new meta-model classes that derive from the abstract classes from **SCE DI** and DI.

The focus of **SDMN DI** is the interchange of laid out shapes and edges that constitute a **SDMN** diagram. Each shape and edge references a particular **SDMN** model element. The referenced **SDMN** model elements are all part of the actual **SDMN** model. As such, **SDMN DI** is meant to only contain information that is neither present nor derivable, from the **SDMN** model whenever possible. Simply put, to render a **SDMN** diagram both the **SDMN DI** instance(s) and the referenced **SDMN** model are REQUIRED.

From the **SDMN DI** perspective, a **SDMN** diagram is a particular snapshot of a **SDMN** model at a certain point in time. Multiple **SDMN** diagrams can be exchanged referencing model elements from the same **SDMN** model. Each diagram may provide an incomplete or partial depiction of the content of the **SDMN** model. As described in Clause 13, a **SDMN** model package consists of one or more files. Each file may contain any number of **SDMN** diagrams. The exporting tool is free to decide how many diagrams are exported and the importing tool is free to decide if and how to present the contained diagrams to the user.

16.3 SDMN Diagram Interchange Meta-Model

16.3.1 How to read this Chapter

Clause 16.4 describes in detail the meta-model used to keep the layout and the look of **SCE** Diagrams (as a basis for **SDMN** Diagrams). Clause 16.4.4 presents in tables a library of the **SCE** element depictions and an unambiguous resolution between a referenced **SDMN** model element and its depiction. Clause 16.5 describes in detail the meta-model used to keep the layout and the look of **SDMN** Diagrams. Clause 16.5.4 presents in tables a library of the **SDMN** element depictions and an unambiguous resolution between a referenced **SDMN** model element and its depiction.

16.3.2 Overview

The **SDMN DI**, which extends the **SCE DI**, is an instance of the OMG **DI** meta-model. The basic concept of **SDMN DI**, as with diagram interchange in general, is that serializing a diagram [*SDMNDiagram*] for interchange requires the specification of a collection of shapes [*SDMNShape*] and edges [*SDMNEdge*].

The **SDMN DI** classes only define the visual properties used for depiction. All other properties that are REQUIRED for the unambiguous depiction of the **SDMN** element are derived from the referenced **SDMN** element [*SDMNElementRef*].

SDMN diagrams may be an incomplete or partial depiction of the content of the **SDMN** model. Some **SDMN** elements from a **SDMN** model may not be present in any of the diagram instances being interchanged.

SDMN DI does not directly provide for any containment concept. The *SDMNDiagram* is an ordered collection of mixed *SDMNShape*(s) and *SDMNEdge*(s). The order of the *SDMNShape*(s) and *SDMNEdge*(s) inside a *SDMNDiagram* determines their Z-order (i.e., what is in front of what). *SDMNShape*(s) and *SDMNEdge*(s) that are *SDMNEdge*(s) MUST appear after them in the *SDMNDiagram*. Thus, the exporting tool MUST order all *SDMNShape*(s) and *SDMNEdge*(s) such that the desired depiction can be rendered.

16.3.3 Measurement Unit

As per OMG DD, all coordinates and lengths defined by **SDMN DI** are assumed to be in user units, except when specified otherwise. A user unit is a value in the user coordinate system, which initially (before any transformation is applied) aligns with the device's coordinate system (for example, a pixel grid of a display). A user unit, therefore, represents a logical rather than physical measurement unit. Since some applications might specify a physical dimension for a diagram as well (mainly for printing purposes), a mapping from a user unit to a physical unit can be specified as a diagram's resolution. Inch is chosen in this specification to avoid variability, but tools can easily convert from/to other preferred physical units. Resolution specifies how many user units fit within one physical unit (for example, a resolution of 300 specifies that 300 user units fit within 1 inch on the device).

16.3.4 Elements

The following sections define the elements necessary for exchanging the diagrams from an **SDMN** modeling tool.

16.3.4.1 SDMNDI

The class SDMNDI is a container for the shared SCE:SCEStyle and all the SDMNDiagrams defined in a SharedDataModel.

The following figure shows the *SDMNDI* metamodel diagram.

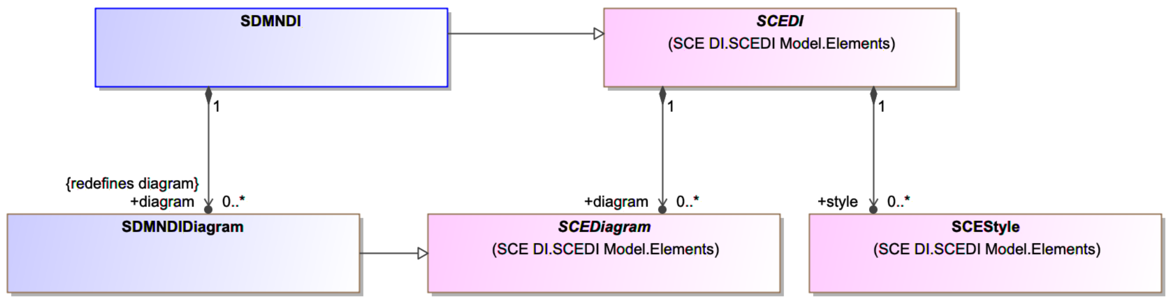


Figure 36: SDMNDI

Generalizations

The *SDMNDI* element does not inherit any attributes or associations of from another element.

Properties

The following table presents the additional attributes and/or associations for *SDMNDI*:

Table 35. SDMNDI Attributes and/or Associations

Property/Association	Description
diagram : SDMNDiagram [0..*]	A list of *SDMNDiagrams*.

16.3.4.2 SDMNDiagram

The class *SDMNDiagram* specializes SCE:SCEDiagram, which specializes DI::Diagram. It is a kind of Diagram that represents a depiction of all or part of a *SDMNDiagram*.

SDMNDiagram is the container of *SDMNDiagramElement* (*SDMNShape*(s) and *SDMNEdge*(s)). *SDMNDiagram* cannot include other *SDMNDiagrams*.

A *SDMNDiagram* can define a SCE:SCEStyle locally and/or it can refer to a shared one defined in the **SDMNDI**.

Properties defined in the local style overrides the one in the referenced shared style. That combined style (shared and local) is the default style for all the *SDMNDiagramElement* contained in this *SDMNDiagram*.

The *SDMNDiagram* class represents a two-dimensional surface with an origin of (0, 0) at the top left corner. This means that the x and y axes have increasing coordinates to the right and bottom. Only positive coordinates are allowed for diagram elements that are nested in a *SDMNDiagram*.

The following figure shows the *SDMNDiagram* metamodel diagram.

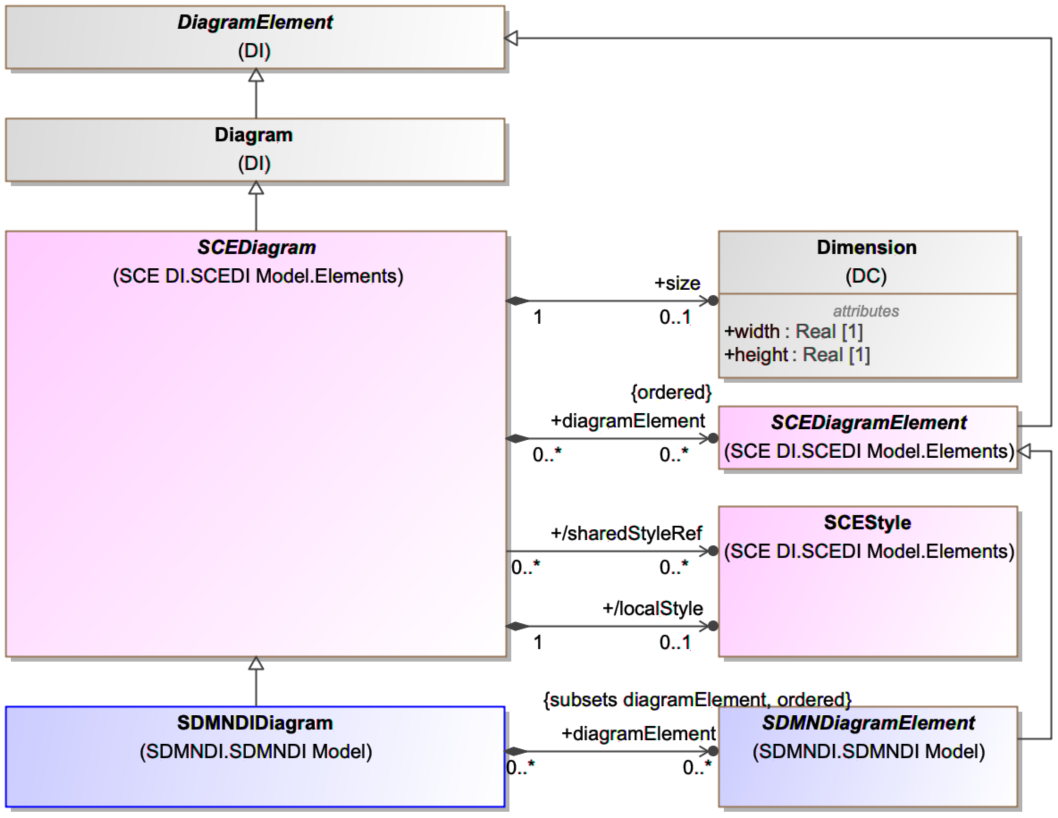


Figure 37: SDMNDI Diagram

Generalizations

The *SDMNDiagram* element inherits the attributes and/or associations of:

- *SCEDiagram* (see the **SCE** specification for more information [OMG doc number bmi-2021-12-09]).

Further, the *SCEDiagram* element inherits the attributes and/or associations of:

- *Diagram* (see the **SCE** specification for more information).

Further, the *Diagram* element inherits the attributes and/or associations of:

- *DiagramElement* (see the **SCE** specification for more information).

In addition, the *DiagramElement* element inherits the attributes and/or associations of:

- *Diagram* (see the **SCE** specification for more information).

Further, the *Diagram* element inherits the attributes and/or associations of:

- *DiagramElement* (see the **SCE** specification for more information).

Properties

The following table presents the additional attributes and/or associations for *SDMNDiagram*:

Table 36. SDMNDiagram Attributes and/or Associations

Property/Association	Description
diagramElement : SDMNDiagramElement [0..*]	A list of SDMNDiagramElements (SDMNShape and SDMNEdge) that are depicted in the SDMNDiagram. This redefines the `diagramElement` association within the SCE:SCEDiagram element.

16.3.4.3 SDMNDiagramElement

The *SDMNDiagramElement* class is contained by the *SDMNDiagram* and is the base class for *SDMNShape* and *SDMNEdge*.

SDMNDiagramElement inherits its styling from its parent *SCEDiagram*. In addition, it can refer to one of the shared SCE:SCEStyle defined in the **SDMNDI** and/or it can define a local style. See section below for more details on styling.

SDMNDiagramElement MAY also contain a SCE:SCELabel when it has a visible text label. If no SCE:SCELabel is defined, the *SDMNDiagramElement* should be depicted without a label.

The following figure shows the *SDMNDiagramElement* metamodel diagram.

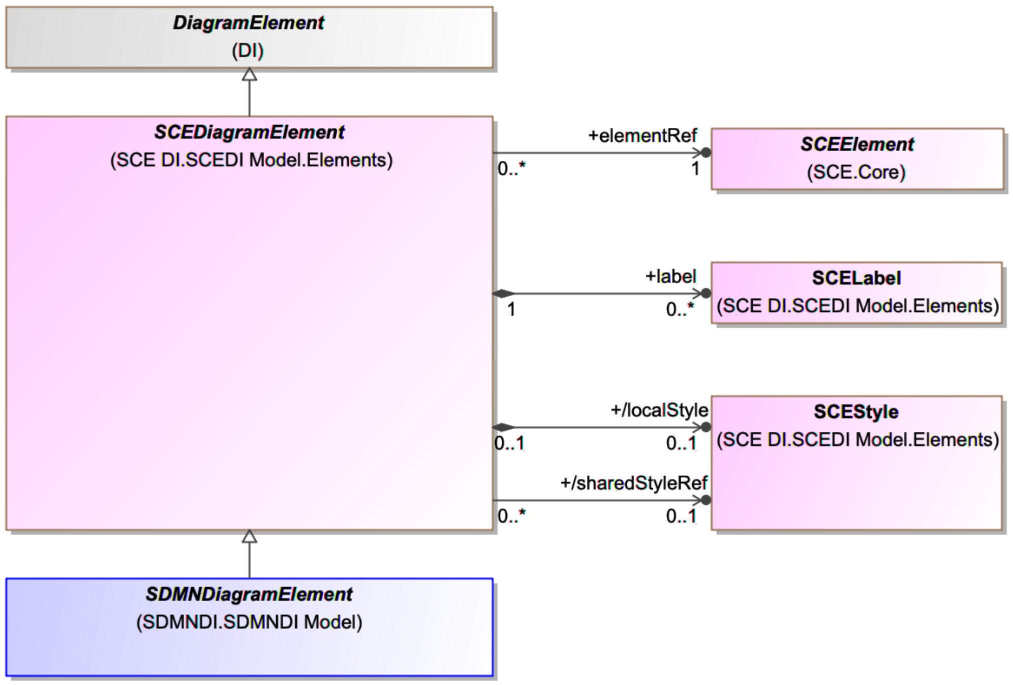


Figure 38: SDMNDI DiagramElement

Generalizations

The *SDMNDiagramElement* element inherits the attributes and/or associations of:

- *SCEDiagramElement* (see the **SCE** specification for more information [OMG doc number bmi-2021-12-09]).

Further, the *SCEDiagramElement* element inherits the attributes and/or associations of:

- *DiagramElement* (see the **SCE** specification for more information).

In addition, the *DiagramElement* element inherits the attributes and/or associations of:

- *DiagramElement* (see the **SCE** specification for more information).

Properties

The *SDMNDiagramElement* element does not have any additional attributes and/or associations.

16.3.4.4 SDMNShape

The *SDMNShape* class specializes SCE:SCEShape, which specializes DI::Shape and SCE:SCEDiagramElement. It is a kind of shape that depicts a *SCEElement* from the *SDMNDiagram* model.

SDMNShape represents a **DataItem** or a SCE DiagramArtifacts **Group** or a **Text Annotation** that is depicted on the diagram. *SDMNDiagram* models may add additional shapes to their diagrams.

SDMNShape has no additional properties but a *SDMNDiagram* model may extend this class to add properties that are used to further specify the appearance of some shapes that cannot be deduced from the *SDMNDiagram* model.

The following figure shows the *SDMNShape* metamodel diagram.

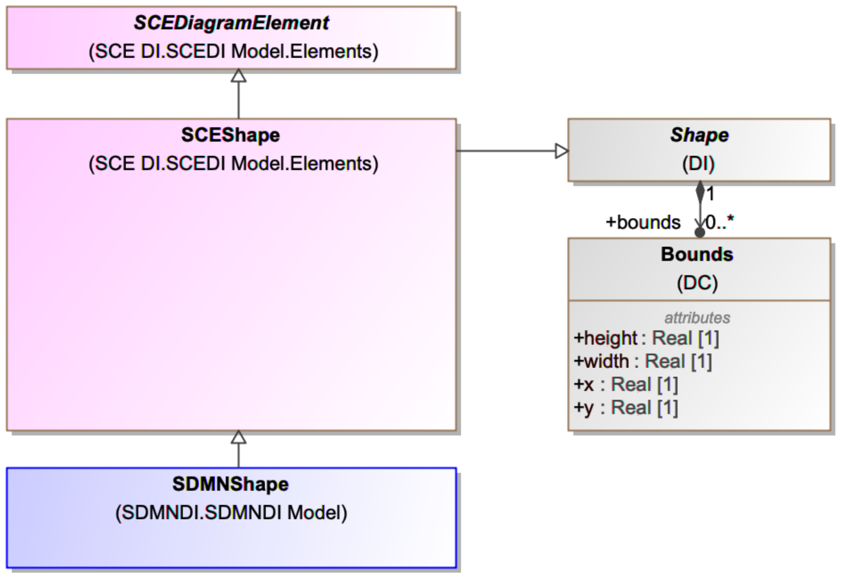


Figure 39: SDMNDI Shape

Generalizations

The *SDMNShape* element inherits the attributes and/or associations of:

- *SCEShape* (see the **SCE** specification for more information [OMG doc number bmi-2021-12-09]).

Further, the *SCEShape* element inherits the attributes and/or associations of:

- *SCEDiagramElement* (see the the **SCE** specification for more information).

Further, the *SCEDiagramElement* element inherits the attributes and/or associations of:

- *DiagramElement* (see the **SCE** specification for more information).

In addition, the *DiagramElement* element inherits the attributes and/or associations of:

- *DiagramElement* (see the **SCE** specification for more information).

In addition, the *DiagramElement* element inherits the attributes and/or associations of:

- *Shape* (see the **SCE** specification for more information).

In addition, the *Shape* element inherits the attributes and/or associations of:

- *Shape* (see the section entitled "Shape" for more information).

In addition, the *SDMNShape* element inherits the attributes and/or associations of:

- *SDMNDiagramElement* (see the section entitled "SDMNDiagramElement" for more information).

Further, the *SDMNDiagramElement* element inherits the attributes and/or associations of:

- *SCEDiagramElement* (see the section entitled "SCEDiagramElement" for more information).

Further, the *SCEDiagramElement* element inherits the attributes and/or associations of:

- *DiagramElement* (see the section entitled "DiagramElement" for more information).

In addition, the *DiagramElement* element inherits the attributes and/or associations of:

- *DiagramElement* (see the section entitled "DiagramElement" for more information).

Properties

The *SDMNShape* element does not have any additional attributes and/or associations.

16.3.4.5 SDMNEdge

The *SDMNEdge* class specializes SCE:SCEEdge, which specializes DI::Edge and SCE:SCEDiagramElement. It is a kind of edge that can depict a relationship between two *SDMNDiagram* model elements.

SDMNEdge are used to depict *Connectors* or **Associations** in the *SDMNDiagram* model. Since *SDMNDiagramElement* might be depicted more than once, `sourceElement` and `targetElement` attributes allow to determine to which depiction a *SDMNEdge* is connected. When *SDMNEdge* has a source, its `sourceModelElement` SHALL refer to the *SDMNDiagramElement* it starts from. That *SDMNDiagramElement* SHALL resolved to the SCE:SCEElement that is the actual source of the *Connector* or **Association**. For Requirement, this is the required SCE:SCEElement. When it has a target, its `targetModelElement` SHALL refer to the *SCEDiagramElement* where it ends. That *SDMNDiagramElement* SHALL resolved to the SCE:SCEElement that is the actual target of the *Connector* or **Association**.

The following figure shows the *SDMNEdge* metamodel diagram.

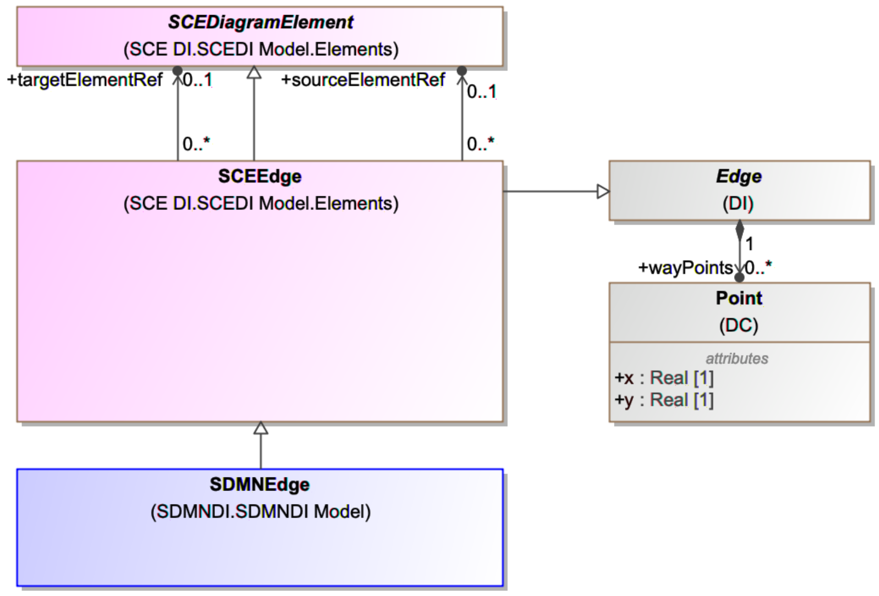


Figure 40: SDMNDI Edge

Generalizations

The *SDMNEdge* element inherits the attributes and/or associations of:

- *SCEEdge* (see the **SCE** specification for more information [OMG doc number bmi-2021-12-09]).

Further, the *SCEEdge* element inherits the attributes and/or associations of:

- *Edge* (see the **SCE** specification for more information).

In addition, the *Edge* element inherits the attributes and/or associations of:

- *Edge* (see the **SCE** specification for more information).

In addition, the *Edge* element inherits the attributes and/or associations of:

- *SCEDiagramElement* (see the **SCE** specification for more information).

Further, the *SCEDiagramElement* element inherits the attributes and/or associations of:

- *DiagramElement* (see the **SCE** specification for more information).

In addition, the *DiagramElement* element inherits the attributes and/or associations of:

- *DiagramElement* (see the **SCE** specification for more information).

In addition, the *SDMNEdge* element inherits the attributes and/or associations of:

- *SDMNDiagramElement* (see the **SCE** specification for more information).

Further, the *SDMNDiagramElement* element inherits the attributes and/or associations of:

- *SCEDiagramElement* (see the **SCE** specification for more information).

Further, the *SCEDiagramElement* element inherits the attributes and/or associations of:

- *DiagramElement* (see the **SCE** specification for more information).

In addition, the *DiagramElement* element inherits the attributes and/or associations of:

- *DiagramElement* (see the **SCE** specification for more information).

Properties

The *SDMNEdge* element does not have any additional attributes and/or associations.

16.3.5 Notation

As a specification that contains notation, **SDMN** specifies the depiction for **SDMN** diagram elements, including **SCE** *DiagramArtifact* elements [OMG doc number bmi-2021-12-09].

Serializing a **SDMN** diagram for interchange requires the specification of a collection of *SDMNShape*(s) and *SDMNEdge*(s) in the *SDMNDiagram* (see sections above). The *SDMNShape*(s) and *SDMNEdge*(s) attributes must be populated in such a way as to allow the unambiguous rendering of the **SDMN** diagram by the receiving party. More specifically, the *SDMNShape*(s) and *SDMNEdge*(s) MUST reference **SDMN** model elements. If no *SCEElement* is referenced or if the reference is invalid, it is expected that this shape or edge should not be depicted.

When rendering a **SDMN** diagram, the correct depiction of a *SDMNShape* or *SDMNEdge* depends mainly on the referenced **SDMN** model element and its particular attributes and/or references. The purpose of this clause is to: provide a library of the **SDMN** element depictions, and to provide an unambiguous resolution between the referenced **SDMN** model element [*SCEElement*] and their depiction. Depiction resolution tables are provided below for both *SDMNShape* and *SDMNEdge*.

16.3.5.1 Labels

Both *SDMNShape* and *SDMNEdge* may have labels (its name attribute) placed on the shape/edge, or above or below the shape/edge, in any direction or location, depending on the preference of the modeler or modeling tool vendor.

Labels are optional for *SDMNShape* and *SDMNEdge*. When there is a label, the position of the label is specified by the bounds of the *SDMNLabel* of the *SDMNShape* or *SDMNEdge*. Simply put, label visibility is defined by the presence of the *SDMNLabel* element.

The bounds of the *SDMNLabel* are optional and always relative to the containing *SDMNDiagram's* origin point. The depiction resolution tables provided below exemplify default label positions if no bounds are provided for the *SDMNLabel* (for *SDMNShape* kinds and *SDMNEdge* kinds (see sections above)).

When the *SDMNLabel* is contained in a *SDMNShape*, the text to display is the name of the *SCEElement*.

16.3.5.2 SDMNShape Resolution

SDMNShape can be used to represent a **Text Annotation** or a **Group**.

DataItems

A **DataItem** is represented in a **SDMN** Diagram as one of two possible shapes.

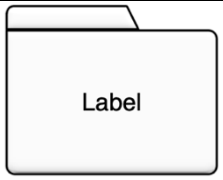
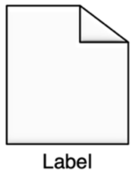
If the **DataItem**'s type is a Folder, then it is displayed with a folder shape (see table below, first row). These **DataItems** are will only be used within a **CMMN** diagram (outside of a **SDMN** diagram) although they will be displayed like any other CaseFileItem.

If the **DataItem**'s type is not a Folder (i.e., any other type of **DataItem**), then it is displayed with a document shape (see table below, second row). These type of **DataItems** can be used in **BPMN**, **CMMN**, and **DMN** diagrams. The **SDMN** shape for these **DataItems** match the shape used in **BPMN** and **CMMN**, but **DMN** uses a lozenge shape for its Data Inputs.

> *Note that the **DataItem** type is determined by the **DataItem**'s associated ItemDefinition.*

The following table presents the depiction resolutions for **DataItems**:

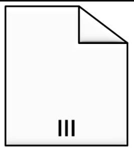
Table 37. **Depiction Resolution of DataItems**

SDMNElement	SDMNShape Attributes	Depiction
DataItem (Folder)	Shapes of **DataItem** that have itemKind=Folder	
DataItem (not Folder)	Shapes of **DataItem** that have itemKind!=Folder	

Multiplicity Decorator

The following table presents the depiction resolutions for the Multiplicity Decorator:

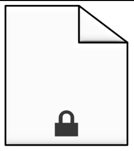
Table 38. **Multiplicity Decorator Depiction**

DataItem Attribute	Depiction
Multiplicity = ZeroOrMore or OneOrMore	

Locked Decorator

The following table presents the depiction resolutions for the Locked Decorator:

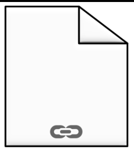
Table 39. **Locked Decorator Depiction**

DataItem Circumstance	Depiction
	

Semantic Reference Decorator

The following table presents the depiction resolutions for the Semantic Reference Decorator:

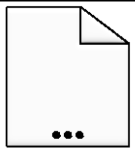
Table 40. **Semantic Reference Decorator Depiction**

DataItem Attribute	Depiction
If the **DataItem** or any component of its associated *ItemDefinition* has an defined *SemanticReference*.	

Hidden Relationship Decorator

The following table presents the depiction resolutions for the Hidden Relationship Decorator:

Table 41. Hidden Relationship Decorator Depiction

DataItem Circumstance	Depiction
	

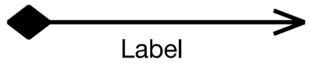
16.3.5.3 SDMNEdge Resolution

SDMNEdge can be used to represent an Ownership Connector, Parent-Child Connector, Relationship Connector, or a Data Association.

CompositionConnector

The following table presents the depiction resolutions for an **CompositionConnector**:

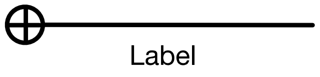
Table 42. Depiction Resolution of the CompositionConnector

SDMN Element	Depiction
CompositionConnector	

ContainmentConnector

The following table presents the depiction resolutions for an **ContainmentConnector** :

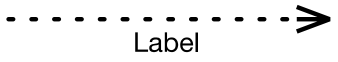
Table 43. Depiction Resolution of the ContainmentConnector

SDMN Element	Depiction
ContainmentConnector	

DataAssociation

The following table presents the depiction resolutions for an **DataAssociation**:

Table 44. Depiction Resolution of DataAssociation

SDMN Element	Depiction
DataAssociation	

ReferenceConnector

The following table presents the depiction resolutions for an **ReferenceConnector**:

Table 45. Depiction Resolution of the ReferenceConnector

SDMN Element	Depiction
ReferenceConnector