



SpecIF

Specification Integration Facility



Gesellschaft für
Systems Engineering e.V.
GERMAN CHAPTER OF INCOSE



SpecIF

Specification Integration Facility

Specification Integration Facility (SpecIF)

Public Released Specification

Version 1.1

2021-11-10

Gesellschaft für Systems Engineering e.V.

Working Group PLM4MBSE

Hermann-Köhl-Straße 7
D-28199 Bremen (Germany)

E-Mail: office@gfse.de

Internet: www.gfse.de or www.gfse.org

CONTRIBUTORS:

Dr.-Ing. Oliver Alt, KARL MAYER STOLL R&D GmbH, Obertshausen

Dr.-Ing. Oskar von Dungern, enso managers GmbH, Berlin

M.Sc. Oliver Eichmann, TUHH Institut für Flugzeug-Kabinensysteme, Hamburg

Dipl.-Math. Uwe Kaufmann, ModelAlchemy Consulting, Falkensee

Nicholas McHardy, KARL MAYER STOLL R&D GmbH, Obertshausen

Dipl.-Ing. (FH) Winfried Reichardt, Green IT Concepts, Rimbach

M.Sc. Steffen Rüscher, TUHH Institut für Flugzeug-Kabinensysteme, Hamburg

Copyright © 2021 Gesellschaft für Systems Engineering e.V. – German Chapter of INCOSE

LICENSE

Licensed under the Apache License, Version 2.0 (the "License");

You may not use this file except in compliance with the License.

You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

Table of contents

Preface	10
1 Introduction	12
1.1 Motivation	12
1.2 Goals	13
1.3 Use Cases	14
2 Concepts	16
2.1 Product Lifecycle Management	16
2.2 Data exchange and data integration in PLM	17
2.3 Application scenarios	18
2.3.1 Data exchange	18
2.3.2 SpecIF as tool back-end for PLM tools	18
2.4 Reuse of existing concepts and standards	18
2.5 SpecIF data is graph data	19
2.6 View concept	20
2.7 Data and Metadata - Classes and Data Types	20
2.8 Inheritance	21
2.9 Data formats	22
2.9.1 JSON	22
2.9.2 JSON Schema	22
2.9.3 File extensions	22
2.9.4 XML representation	23
2.9.5 SpecIF diagram interchange	23
2.10 Web API	24
2.11 Multilingualism	24
2.12 Versioning	24
2.13 Semantics	26
2.14 Semantic model-data integration using the Fundamental Modeling Concepts	26
2.15 Model Integration Guides	28
2.16 Versioning of this specification	29
3 SpecIF Metamodel	30
3.1 SpecIF Repositories	31
3.1.1 SpecIF-Repository description attributes	32
3.2 Data representation and data type definitions in SpecIF	33

3.3	Metamodel helper classes	34
3.3.1	Base Element	34
3.3.2	MultilanguageText	35
3.3.3	EnumeratedValue	35
3.3.4	AlternativeId	36
3.4	SpecIF-Metamodel details	37
3.4.1	DataType	38
3.4.2	PropertyClass	39
3.4.3	ResourceClass	39
3.4.4	StatementClass	40
3.4.5	Property	41
3.4.6	Resource	42
3.4.7	Statement	42
3.4.8	Node	43
3.4.9	File	43
4	SpecIF JSON-Schema	46
4.1	SpecIF JSON example	46
4.2	Definition elements and data elements	47
4.3	Definition elements	47
4.3.1	Data types	47
4.3.2	Enumerations	50
4.3.3	Property classes	52
4.3.4	Resource Classes	54
4.3.5	Statement Classes	55
4.4	Data elements	56
4.4.1	Resources	56
4.4.2	Statements	57
4.5	Hierarchies and Nodes	58
4.6	Files	59
5	SpecIF Diagram Exchange using SVG	61
5.1	XML namespaces in SVG	61
5.2	Embedding SpecIF-SVG data into resource elements	62
5.3	Coordinate system	62
5.4	SVG structure and grouping	63
5.5	SpecIF diagram exchange metadata	65

5.5.1	Referencing model elements	65
5.6	The tags specif:shape and specif:edge	66
5.6.1	Diagram metadata	66
5.6.2	Element metadata	66
5.6.3	Connector metadata	67
6	SpecIF Web API	69
6.1	Structure of the SpecIF Web API	69
6.1.1	CRUD operations	69
6.1.2	API versioning	69
6.1.3	Authentication and Authorization	69
6.1.4	API Key Authentication	70
6.1.5	Role-based authorization	70
6.1.6	Error handling	71
6.1.7	Data model and data format	71
6.1.8	Parameters	71
6.2	Handling of revisions	72
6.2.1	Revisioning for data elements	72
6.2.2	Revisioning for meta elements	72
6.3	Data definition endpoints	73
6.3.1	Data types	73
6.3.2	Property classes	73
6.3.3	Resource classes	74
6.3.4	Statement classes	75
6.4	Data endpoints	76
6.4.1	Resources	76
6.4.2	Statements	76
6.4.3	Hierarchies	77
6.4.4	Projects	78
6.4.5	Files	80
7	SpecIF Class Definitions	81
7.1	Domains	81
7.1.1	Domain types	82
7.2	Domain 01: Base Definitions	83
7.2.1	Data types of domain 01: Base Definitions	83
7.2.2	Property classes of domain 01: Base Definitions	83

7.2.3	Resource classes of domain 01: Base Definitions.....	85
7.2.4	Statement classes of domain 01: Base Definitions.....	85
7.3	Domain 02: Requirements Engineering	85
7.3.1	Data types of domain 02: Requirements Engineering.....	85
7.3.2	Property classes of domain 02: Requirements Engineering.....	85
7.3.3	Resource classes of domain 02: Requirements Engineering.....	86
7.3.4	Statement classes of domain 02: Requirements Engineering.....	86
7.4	Domain 03: Model Integration	86
7.4.1	Data types of domain 03: Model Integration	86
7.4.2	Property classes of domain 03: Model Integration.....	86
7.4.3	Resource classes of domain 03: Model Integration	87
7.4.4	Statement classes of domain 03: Model Integration	88
8	Introduction to SpecIF Model Integration.....	90
8.1	Model Integration Resources	90
8.1.1	Fundamental Model Element Types	90
8.1.2	Requirement and Feature	91
8.1.3	View	91
8.1.4	Package.....	92
8.1.5	Collection	92
8.1.6	A glimpse on the elements of SpecIF Model Integration.....	93
8.2	Model Integration statements	93
8.2.1	Expressing structure	94
8.2.2	Expressing traceability aspects.....	96
8.2.3	Expressing behavior	97
8.2.4	Instantiation	97
8.2.5	Document outlines.....	98
8.2.6	Comments.....	98
8.3	SpecIF Classes for Model Integration	99
8.3.1	Mapping of different modeling environments to SpecIF	100
9	SpecIF Model Integration Guide for ArchiMate®	102
9.1	ArchiMate® to SpecIF mapping	102
9.1.1	Resources	102
9.1.2	Statements	102
9.1.3	Example.....	103
9.1.4	Transformation Code.....	105

10	SpecIF Model Integration Guide for BPMN	106
10.1	BPMN to SpecIF mapping	106
10.1.1	Resources	106
10.1.2	Statements	107
10.1.3	Example.....	108
10.1.4	Transformation Code.....	109
11	SpecIF Model-Integration Guide for FMC	110
11.1	FMC to SpecIF Mapping	110
11.1.1	Resources	111
11.1.2	Statements	111
11.1.3	Example.....	112
11.1.4	Transformation Code.....	112
12	SpecIF Model Integration Guide for UML and SysML.....	113
12.1	UML/SysML to SpecIF Mapping.....	113
12.1.1	Mapping of UML Profiles.....	113
12.1.2	Resource mapping tables.....	114
12.1.3	Statement mappings	118
12.1.4	Property mappings.....	119
12.2	Mapping of the model structure.....	120
12.3	Examples	122
12.3.1	Examples for UML/SysML mapping and transformation of activity diagrams 122	
12.3.2	Example for mapping state charts and state transitions to the SecIF Integration Model elements	126
13	SpecIF-ReqIF Mapping	128
13.1	Datatypes	128
13.1.1	Strings.....	128
13.1.2	Boolean.....	128
13.1.3	Byte	128
13.1.4	Integer.....	128
13.1.5	Real.....	129
13.1.6	Date	129
13.1.7	XHTML.....	129
13.1.8	Enumeration	129
13.2	SpecIF schema attributes	130

13.2.1	ResourceClasses	130
13.2.2	StatementClasses	132
13.2.3	Resources	133
13.2.4	Statements	134
13.3	Hierarchies	135
14	Non normative class definitions	137
14.1	Domain 04: Automotive Requirements Engineering	137
14.1.1	Data types of domain 04: Automotive Requirements Engineering.....	137
14.1.2	Property classes of domain 04: Automotive Requirements Engineering.....	137
14.1.3	Resource classes of domain 04: Automotive Requirements Engineering.....	137
14.2	Domain 05: Agile Requirements Engineering.....	138
14.2.1	Resource classes of domain 05: Agile Requirements Engineering.....	138
14.3	Domain 07: Issue Management	138
14.3.1	Data types of domain 07: Issue Management	138
14.3.2	Property classes of domain 07: Issue Management	138
14.3.3	Resource classes of domain 07: Issue Management	138
14.4	Domain 08: BOM	138
14.4.1	Resource classes of domain 08: BOM	138
14.5	Domain 09: Variant Management	139
14.5.1	Data types of domain 09: Variant Management.....	139
14.5.2	Property classes of domain 09: Variant Management.....	139
14.5.3	Resource classes of domain 09: Variant Management.....	139
14.6	Domain 10: Vocabulary Definition	140
14.6.1	Resource classes of domain 10: Vocabulary Definition	140
14.6.2	Statement classes of domain 10: Vocabulary Definition	140
14.7	Domain 11: Testing	141
14.7.1	Data types of domain 11: Testing	141
14.7.2	Property classes of domain 11: Testing.....	141
14.7.3	Resource classes of domain 11: Testing	141
14.7.4	Statement classes of domain 11: Testing	142
14.8	Domain 12: SpecIF Events	142
14.8.1	Data types of domain 12: SpecIF Events	142
14.8.2	Property classes of domain 12: SpecIF Events	142
	Resource classes of domain 12: SpecIF Events	143
15	References	144



15.1	SpecIF publications	144
15.2	Standards	145

Preface

Systems engineering has become an indispensable part of cooperation between companies in the manufacturing, utility, health and other industries. In the whole lifecycle of products, engineers must exchange information in many formats from different sources. For better understanding, the data must be semantically linked. This is where the *Specification Integration Facility* (SpecIF) comes into play, which version 1.1 is described by this specification.

When manufacturers and suppliers want to coordinate the design of a complex product with all its components, the interaction between mechanics, electronics and software must be analyzed and tested. Systems engineering methods are increasingly gaining acceptance; the tools and data used in the respective disciplines must engage with each other in an integrative manner. With model-based systems engineering, the transition from a document-centered to an artifact-based working style is facilitated. SpecIF gathers the information generated in the whole product lifecycle in a semantic net, allowing for high scalability and ultra-fast searching.

SpecIF is method- as well as vendor-independent and is based on international standards such as the Requirements Interchange Format (OMG ReqIF). SpecIF also adopts concepts of the Dublin Core Metadata Initiative (DCMI) for the categorization and description of information items and endeavors to integrate or develop further vocabularies or ontologies. The GfSE strives to establish SpecIF as an international standard and cooperates with the Object Management Group (OMG) and other interest groups.

SpecIF is being developed as an open-source project with a free license including even commercial use, so that users and product providers alike can participate in the initiative and make their own contributions.

The GfSE working group *Product Lifecycle Management for Model-based Systems Engineering* (PLM4MBSE) pursues the goal of digital transformation in product development from classic, geometry- and CAD-centered Product Lifecycle Management to interdisciplinary Model-based Systems Engineering. The members of this working group are the main contributors behind the SpecIF.

SpecIF is a vital GfSE project with promising benefits! I would like to thank the active contributors and wish them a wealth of ideas and creativity for the future!

Walter Koch

Chairman of GfSE e. V. and President German Chapter of INCOSE

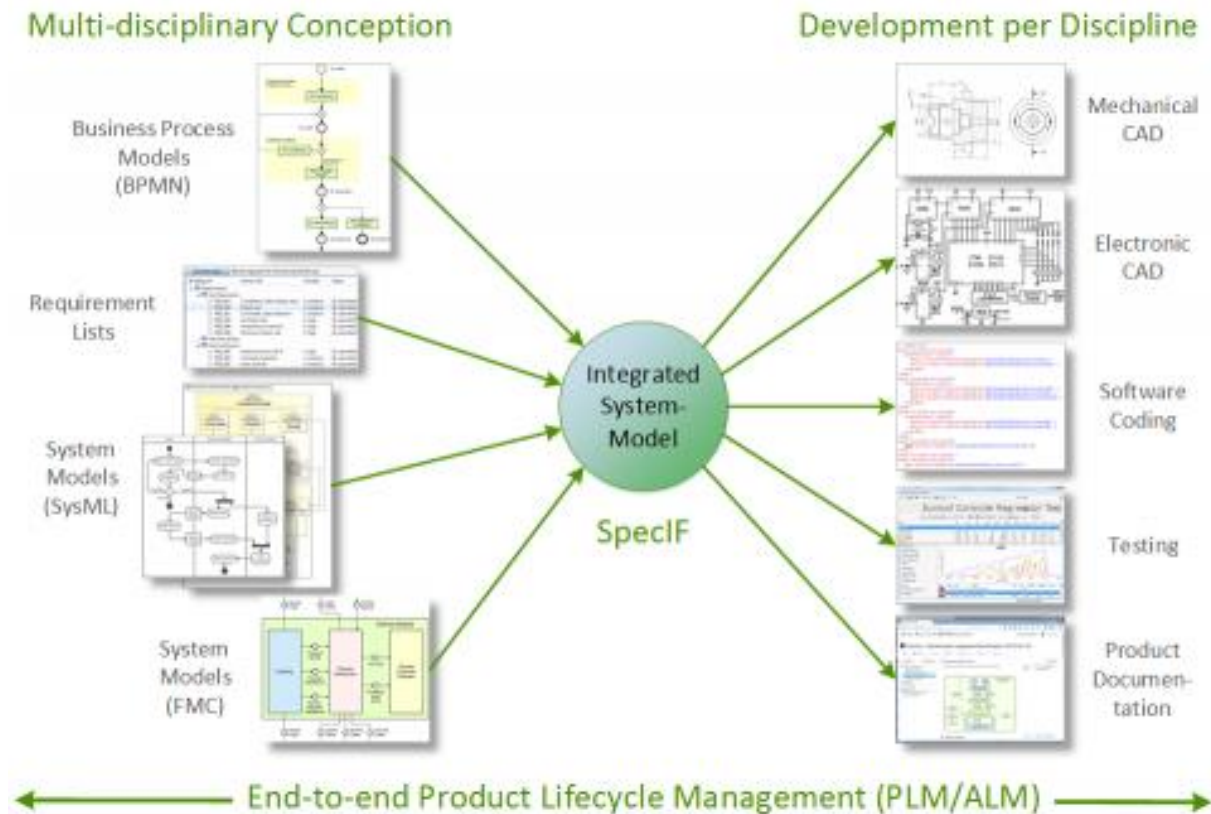
1 Introduction

1.1 Motivation

In the field of Systems Engineering (SE) a multitude of methods is being used with benefit every day; for example, requirements management, modeling of system structure and behavior with UML/SysML or simulation with Modelica and other languages. There is information from various sources and in different formats, all providing valuable input for system design and development.

In practice, it is difficult or even impossible to join the information with acceptable effort and put it in relation. Information from different sources (“silos”) is often inconsistent, because it is maintained by different organizations with their own background and purpose. Popular modeling standards such as UML/SysML are notations but leave semantic interpretation to tool makers or users. For data (model) exchange there are several standards with respect to syntax, but very few which address the semantics as well.

The Specification Integration Facility (SpecIF) shall support the change from document-centric to artifact-centric collaboration, which is a generally accepted goal in the domains of systems engineering and product lifecycle management (PLM). SpecIF defines a language for describing system models with attention to both syntax and semantics. By creating a common context for graphical and textual content, an understanding (beyond mere communication) is achieved on a logical level. Existing technical formats and protocols such as ReqIF or RDF are adopted to take advantage of existing IT infrastructure.



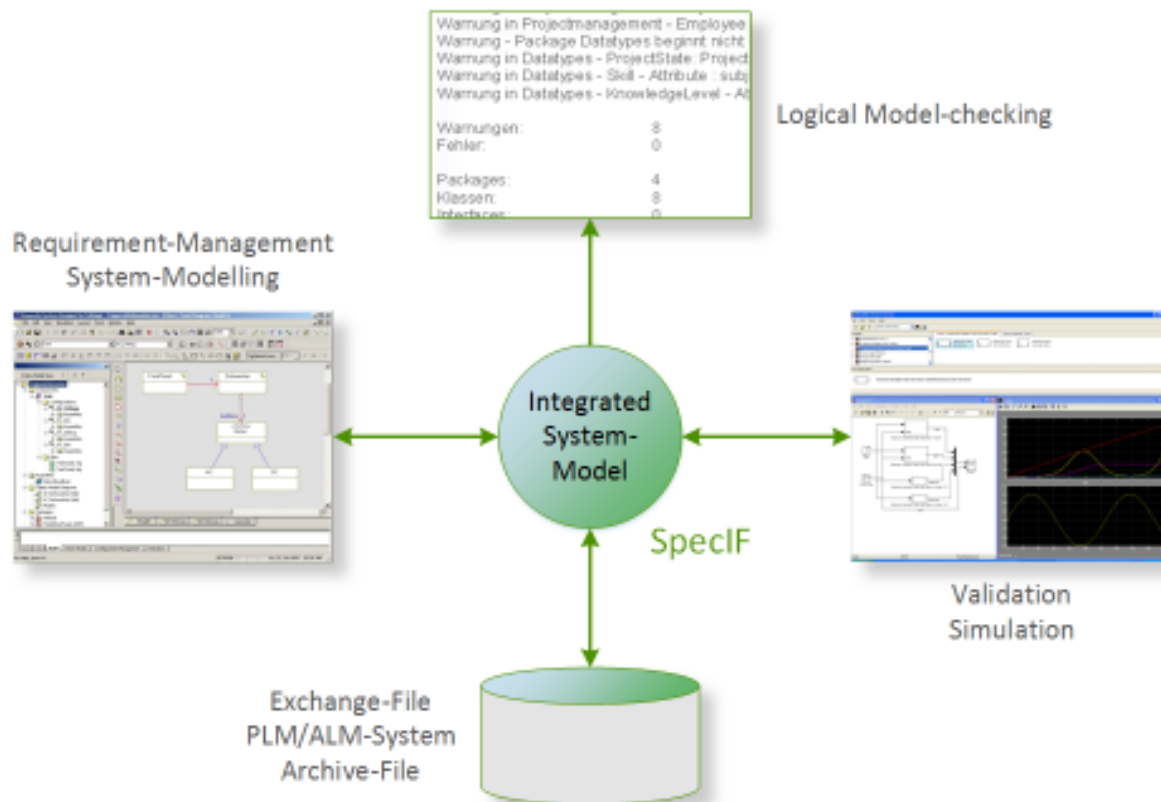
End-to-end product lifecycle management

1.2 Goals

SpecIF contributes to the following objectives:

- Lifecycle-Management from the beginning: Structures and content from the early phases of system conception are seamlessly made available for development.
- Embracing disciplines: SpecIF creates a common context for models from disciplines such as Mechanics, Electronics, Software, Safety and others.
- Embracing methods: Texts as well as structural and behavioral models of popular methods, among others BPMN, SysML and FMC can be integrated. This means that individual elements (“resources”) exist once and may appear on several model diagrams.
- Technology-neutral: SpecIF data can be transformed to various technical formats, such as ReqIF, OSLC, XMI, relational database, graph database or web linked-data (RDF).
- Vendor-neutral and independent: SpecIF is not limited to certain tools or vendors; in contrast, SpecIF lets you exchange model data between different tools and organizations.
- Schema-compliant: SpecIF data can be checked formally using a JSON- or XML-schema; the former has been made available at [SpecIF-Schema](#).

- Standard-compliant: SpecIF draws on existing standards, most importantly from W3C, OMG and OASIS.
- Open and cooperative: All results are published with [Creative Commons 4.0 CC BY-SA license](https://creativecommons.org/licenses/by-sa/4.0/); allowing commercial use. The results can be further developed, but the origin must be stated, and they must be published under similar terms; please consult the referenced license text. We encourage everyone interested to join our GfSE working group and to directly contribute to the results.



End-to-end product lifecycle management

1.3 Use Cases

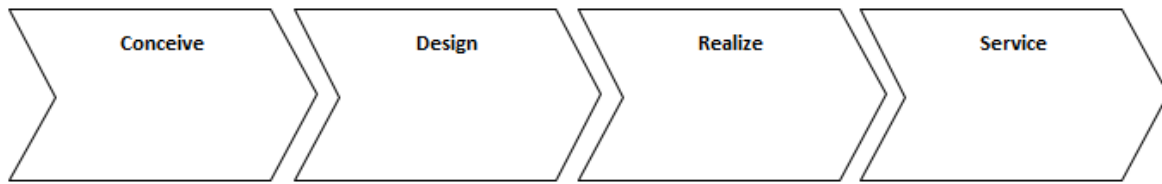
Today, there is close cooperation between product OEMs, engineering service providers and suppliers. The business processes demand easy information exchange between all participating organizations:

- Exchange requirement-specifications and model-based system-specifications along the supply-chain.
- Publish results (including system models) from different authoring systems, usually requiring a high level of expertise, to a far bigger group of ‘occasional’ users for inquiry, commenting or auditing. Uniform access regardless of authoring system is best practice.
- Show changes made over time and support the change management across organizations.

With respect to the content, information comes from different organizations and addresses product strategy and resulting requirements, laws and consumer protection, optimized user interaction, functions, system structure and behavior or even the validation of ratings by means of simulation. The following tasks shall be supported:

- Integrate information and models from different sources and in different formats,
- Search and navigate consistently,
- Find and consolidate identical elements in different models,
- Detect and store dependencies and logical relations between model elements, essentially interrelating model elements with a semantic net,
- Detect errors, inconsistencies or violations of design rules,
- Reference ('trace') artifacts in subsequent development steps without copying.

4. Service phase (Service - Use, operate, maintain, support, sustain, phase-out, retire, recycle and disposal)



The four phases of PLM

In all PLM phases different data is created, edited and exchanged between different stakeholders. SpecIF is defined as the new universal standard for representation, exchange and integration of all kinds of PLM data in all phases.

2.2 Data exchange and data integration in PLM

Data exchange and data integration in a product lifecycle is one of the main tasks in PLM. Nowadays many different tools are typically used to support the different phases of PLM and no common standard is available to exchange and integrate PLM data between different tools.

Some standards are available for single domains like CAD data exchange or requirements exchange. Unfortunately these standards do not cover the entire PLM and often manual work is required when exchanging data between different tools, because of some tool-specific interpretations of exchange standards.

To solve these problems, SpecIF will define a standard for data exchange and integrate PLM data of any kind. To achieve this goal SpecIF defines a syntax for a data format and the semantics, describing how specific PLM data shall be expressed using the SpecIF syntax. Therefore, SpecIF defines

1. A *syntax* for data representation (data format(s)).
2. A definition of (data) type names and their meanings (*semantics*) - sometimes also called *Vocabulary*.
3. Different *application guides*, that describe how to map specific PLM data to SpecIF.
4. A definition of a *Web API* that is able to provide and consume SpecIF data - helpful for data integration scenarios using web technologies.

2.3 Application scenarios

SpecIF can be used in at least two different application scenarios.

2.3.1 Data exchange

It can be used as data exchange format, to exchange PLM data between different tools used in the product life cycle.

A typical example is the discipline of requirements engineering: A requirements engineer wants to provide a specification to a supplier or a customer for review. With SpecIF the requirements engineer can export the specification contents to a SpecIF file and send it to the stakeholder. The stakeholder can then open or import the SpecIF data with his own SpecIF-supporting tools, make some additions or comments and send the result back to the requirements engineer. Because of the clear defined syntax and semantics of SpecIF, no manual work for data mapping or data integration is necessary.

2.3.2 SpecIF as tool back-end for PLM tools

The second application scenario is the usage of SpecIF as direct back-end for PLM tools. SpecIF provides all necessary capabilities (syntax, semantics and the Web API definition) to use it as a tool data back-end. The typical Create, Read, Update and Delete operations (CRUD) [[WikipediaCRUD](#)] are supported which are required for any kind of PLM tool.

Furthermore SpecIF implements the concept of separation of model and view (see section *View concept*) as used in many PLM tools like UML-based modeling platforms or requirements engineering tools.

2.4 Reuse of existing concepts and standards

SpecIF does not want to reinvent the wheel again. Instead, SpecIF reuses concepts and terms of existing and established standards as often as possible.

SpecIF uses selected concepts and terms defined by the following standards:

- The [Dublincore Metadata Initiative](#) (Dublin Core)
- The [Requirements Interchange Format](#) (ReqIF)
- The [International Requirements Engineering Board](#) (IREB)
- The [Fundamental Modeling Concepts](#) (FMC)

- The [Unified Modeling Language](#) (UML) and their dialects (e.g. SysML)
- The [Business Process Model and Notation](#) (BPMN)

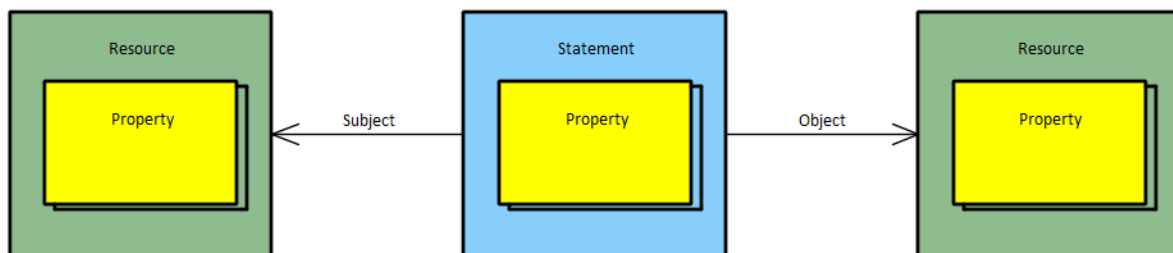
For SpecIF data syntax representation and definition the following existing standards are used:

- JavaScript Object Notation (JSON) and the JSON-Schema for data format specification
- XHTML to represent formatted text
- SVG to represent vector graphic data (used for diagram data exchange)
- Swagger resp. OpenAPI for definition of the SpecIF-WebAPI
- The Meta Object Facility (MOF) meta-modeling and the Model-Driven Architecture (MDA) approaches defined by the [Object Management Group \(OMG\)](#)

2.5 SpecIF data is graph data

All data represented with SpecIF is graph data consisting of nodes and edges. The nodes are called *Resources* in SpecIF terminology, the edges are called *Statements*.

Resources and Statements can have a list of defined *Properties*. Each property is a key/value pair storing a data value.



Principle of SpecIF data representation

A statement is not saved grouped with the resource data. Instead, it has two references using unique identifiers (GUIDs) to define the two connected resources. Each resource element has its own unique identifier.

In a SpecIF statement the resource element where the statement starts is named as the statement *subject* and the resource element where the statement ends is named as the statement *object*.

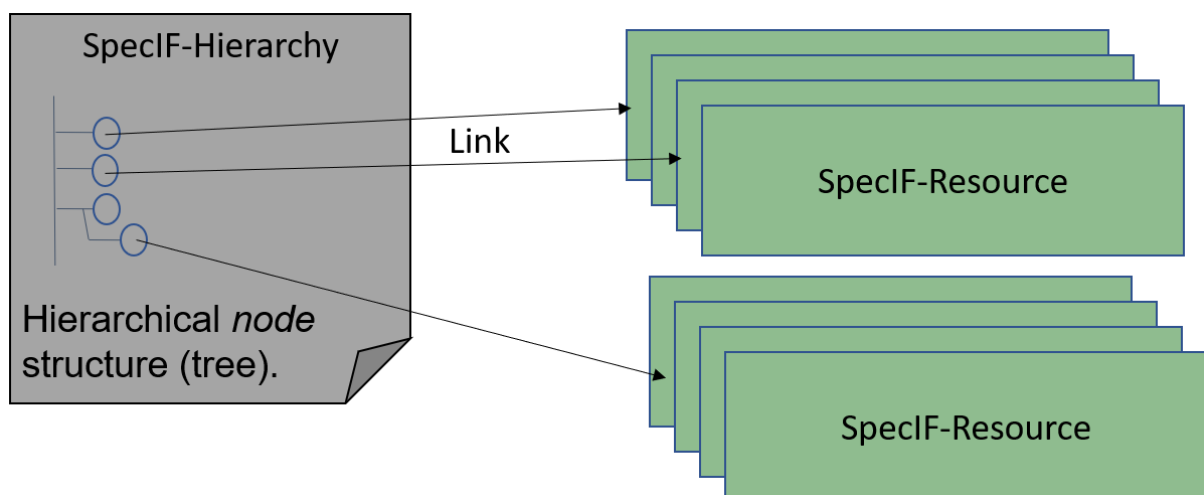
Statements itself can also be subject and/or object of another statement. Therefore, it is possible to create n-ary statements in SpecIF.

2.6 View concept

SpecIF supports the concept of separation of model and view. The model is the collection of existing SpecIF-Resource elements. The view is a tree data structure called *hierarchy* where a selection of the SpecIF-Resource elements are shown in a hierarchical order. The elements of a hierarchy structure in SpecIF are called *nodes*.

It is not required, that all available resources are part of a hierarchy structure. Each node in a hierarchy tree defines a reference to a specific resource element. The resources are linked by the nodes using the resource unique identifier. It is possible to reference the same resource element in multiple hierarchy structures. This allows reuse of existing data without copying the data.

The following figure illustrates the concept:



Principle of the SpecIF view concept

The hierarchy consists of a tree of nodes and each node links to a specific resource element. Typical application scenarios are a document view for a requirement document using hierarchy to provide a chapter and section structure. Another scenario is the representation of a model tree in a UML-tool consisting of UML-packages, UML-diagrams and UML-elements.

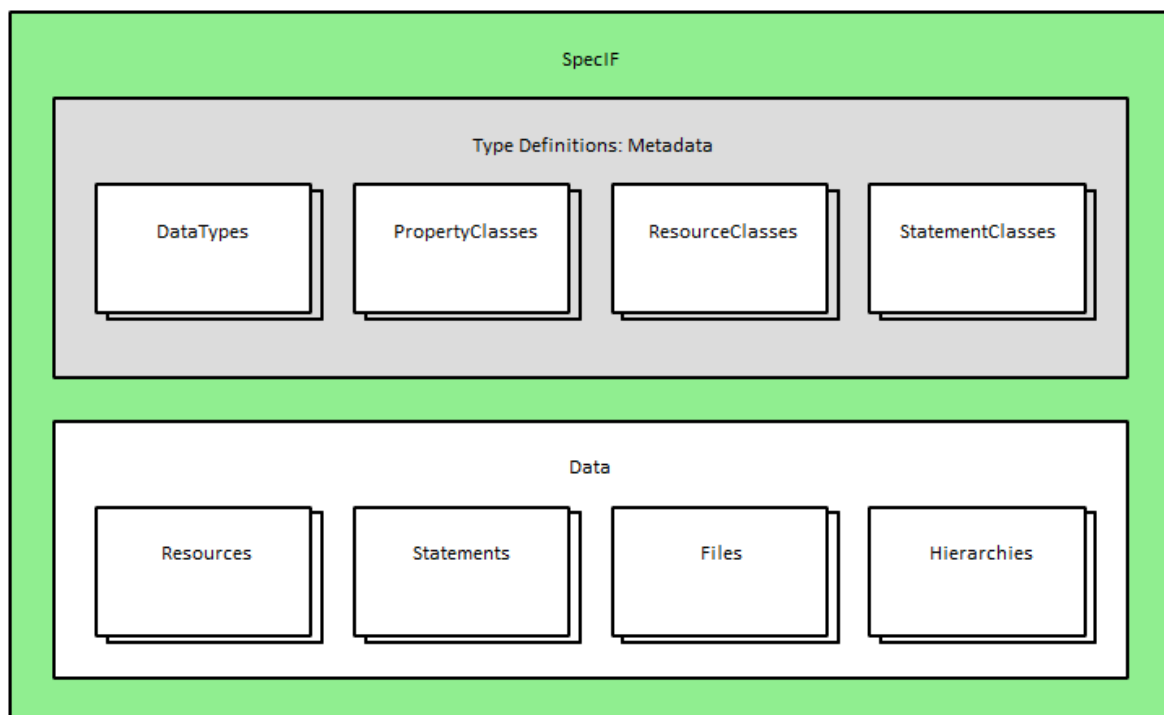
2.7 Data and Metadata - Classes and Data Types

SpecIF allows not just the representation of PLM data, but also provides mechanisms to define how the data is structured and how resources and statements are created to represent a specific kind of data. Each SpecIF resource and statement is defined with SpecIF itself. This principle

is called meta-modeling in the domain of object-oriented data modeling and is also known in the object-oriented software design called the principle of class and object or class and instance.

For each SpecIF data element a defining class element must exist. The SpecIF terms for the data and meta data types (classes) are:

- A *property class* element defines a property type.
- A *resource class* element defines a resource type.
- A *statement class* element defines a statement type.
- A *data type* element defines the data types used in the property type definitions (property classes).



The SpecIF architecture

2.8 Inheritance

SpecIF implements the concept of inheritance - well known from object-oriented software development. It is possible to define a new resource- or statement-class based on an already defined element. The new element (child element) inherits all properties from the element it is derived from (parent element). It is possible to define additional properties on the child element. SpecIF uses the term *extends* to store a reference to the parent element from the child element.

The concept of inheritance in SpecIF is very helpful, when a specific application scenario requires special, additional domain-specific properties. The SpecIF user can define the new classes based on standardized types. A tool, that does not know the application-specific properties, but the standardized base-types, can handle the data as any other standard data with extensions.

2.9 Data formats

SpecIF data is represented in specific data formats following conventions defined by this specification. The following sections describe these concepts.

2.9.1 JSON

SpecIF data uses the [JavaScript Object Notation \(JSON\)](#) to represent its data. JSON is a widely used format to represent object oriented data structures. It supports the data representation of data objects including arrays.

JSON is selected as data format for SpecIF, because it offers

- very good tool support on all platforms,
- a high level of familiarity,
- a more compact data representation in comparison to XML,
- a superior and easy-to-use support for schema definition and schema validation,
- the possibility to use it as file-based or Web API based format in the same way.

2.9.2 JSON Schema

JSON can represent any kind of object-oriented data object. To define the syntax of SpecIF, [JSONSchema](#) is used. JSON Schema was introduced to define data formats expressed with JSON.

JSON Schema itself uses a special kind of JSON format to define any JSON format such as SpecIF.

2.9.3 File extensions

SpecIF data can be persisted to a JSON file using one of the following methods:

- The SpecIF file gets the file extension *.specif* and contains the JSON serialization of the SpecIF data.
- SpecIF also supports the inclusion of any other files (e.g. images, PDF documents etc.) referenced by a SpecIF content. These attached files are stored together with the *.specif*

file inside a ZIP-archive with the file extension *.specifz*. One or more *.specif file are contained at root level of the zipped archive and the referenced files are often collected in a folder; sometimes named *files_and_images*.

The following table shows an overview about SpecIF file extensions:

File extension	Meaning
.specif.json or .specif	SpecIF file with JSON content defined by the SpecIF-JSON-Schema.
.specif.xml or .specifx	Reserved for future use to store SpecIF data saved as XML.
.specif.zip or .specifz	SpecIF zipped archive with at least one *.specif (or *.specifx) file at root level and the referenced files, often in a folder <i>files_and_images</i> .
.specif.html	HTML with embedded SpecIF data.

2.9.4 XML representation

SpecIF v1.1 does not define a XML resp. RDF representation. Nevertheless, it may be provided in future releases.

2.9.5 SpecIF diagram interchange

A diagram is a graphical representation of graph-based data. Examples for such diagrams are electrical schematics, UML, SysML or FMC diagrams or any other diagrams showing graph data consisting of nodes and edges. In SpecIF the data behind the graph nodes are the Resources and the data behind the edges are the Statements.

The simplest way to include graphical diagrams in SpecIF-data is the usage of graphics in existing formats (e.g. PNG, GIF, TIFF etc.) and include them into a XHTML property as simple image.

To semantically integrate diagram data in SpecIF, the Scalable Vector Graphics (SVG) standard is used (<https://www.w3.org/TR/SVG2/>). SVG is an XML-standard, defined by the W3C, used to define scalable 2D vector graphics. The standard defines a wide range of possibilities to define vector-based graphics. SVG files can be opened and viewed with all modern web browsers and SVG allows the inclusion of meta-data extensions into the SVG file.

This is the main reason why SVG is selected as standard for SpecIF diagram exchange, because SpecIF uses SVG and extends the graphical data by meta information for semantic diagram exchange. The resulting SpecIF-compliant SVG contains the graphical information, that can be

used with all SVG viewers, but it also contains some semantic meta-information that allows traceability to SpecIF resources and statements, visualized in the SVG.

The concrete definition of the SpecIF-SVG-metadata is defined in a separate chapter of this specification.

2.10 Web API

Besides the possibility to store SpecIF data in a file, the data may be persisted using other physical storage options like SQL- and NoSQL-databases. To provide a common access point for all SpecIF data a Web API definition is part of SpecIF. The Web API definition uses Swagger/OpenAPI technologies to define endpoints and data models to create, read, update and delete (CRUD operations) SpecIF data.

2.11 Multilingualism

In the product lifecycle projects and products with international development teams, international markets and customers must often be addressed. Therefore SpecIF supports data representation in different languages in parallel. This multilingualism of SpecIF is of course optional.

For this purpose all property values of type *xs:string* are a list with the object specifying the text itself, it's language and it's format. Any user or hosting system may select the language to show.

The preferred language may be specified at project, resource, statement or property level. A more granular definition supersedes a more general one. If no preferred language is specified, 'en' is assumed.

2.12 Versioning

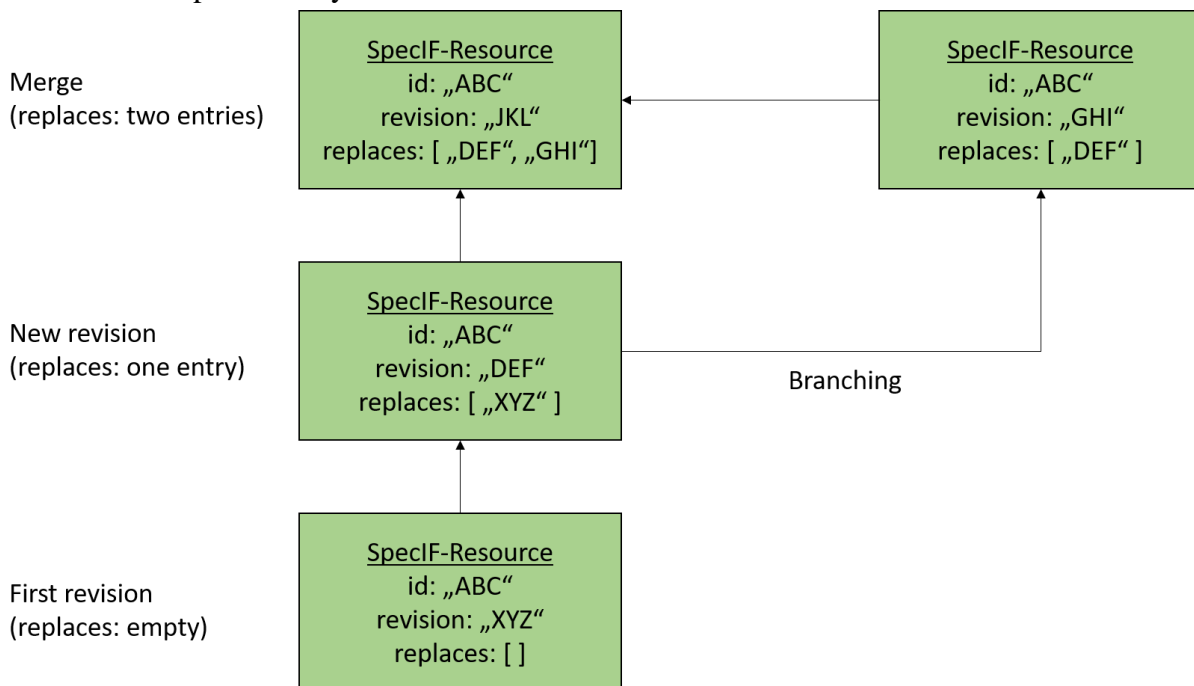
SpecIF supports versioning for all kinds of SpecIF data like data types and class definitions as well as resource and statement data. It is not required to store SpecIF files in a separate version control system (e.g. Git or SVN), because SpecIF includes versioning concepts and each element has revision information.

If a change for a SpecIF element is initiated and the used tool supports the versioning concept of SpecIF, the chosen element is not changed. Instead a copy of the chosen element is created

and then the change is applied to this copy. The copy element is saved as a revision with the same element ID, but a different revision ID. The new element revision must include a reference to the revision ID of the original element.

This leads to the following SpecIF-versioning rules:

- Multiple revisions of a SpecIF element have the same element ID, but each revision must have a different unique *revision ID*.
- Each SpecIF element shall have zero, one or two entries with a revision ID in the JSON-property *replaces*.
- If an element has no replaces entry, it is the first revision.
- If an element has one replaces entry, it is the successor of the revision given in the replaces entry.
- If an element has two replaces entries, it is the merge result of the two elements given in the replaces entry.



Versioning in SpecIF

The figure above shows an example of the different versioning and revision scenarios in SpecIF using an example of a resource element. The resource element has the ID “ABC”. Each revision of this resource has the same ID but different revision IDs and replaces references.

With this concept it is possible to support the concepts of linear revisioning, branching, and merging. A typical application scenario is a data export of PLM data from a source data

provider, parallel changes from multiple stakeholders and the reintegration of the data into the source data provider.

2.13 Semantics

To avoid mapping data manually when exporting and importing data in different tools, the SpecIF standard defines standardized naming (*title*) for data representations and their meaning (semantics). Most names come from the Dublin Core Metadata Initiative.

One example: The Dublin Core Metadata Initiative has defined the term *dcterms:title* to represent any kind of a name. Therefore, in SpecIF a propertyClass named *dcterms:title* is used to assert an element's title.

In every application domain all properties containing a name use the term *dcterms:title* as key for this property. It is used in a requirement to represent the requirement title just as in a SpecIF-UML-mapping to represent the name of a class or any other UML model element and so on.

Using a standardized SpecIF definition of such terms in different PLM application domains makes it possible to exchange data between all tools supporting the SpecIF standard without the effort of manual mapping data types and property types when exporting or importing data to or from SpecIF.

2.14 Semantic model-data integration using the Fundamental Modeling Concepts









Few methods span all required aspects of a complex system. In large organizations, business process management, IT architecture management and requirement management, all being recognized fields of specialization and expertise, use different methods and tools. The fact is equally apparent in mechatronic system development, where mechanical, electrical and software engineers are designing a common system with their respective methods. Design alternatives should be discussed and evaluated in a common effort of all disciplines. But how to document the results? An overarching modeling approach is needed: Different models must be semantically integrated.

It is no viable approach to ask all stakeholders to use the same method or even the same tool. Our approach keeps the proven methods and tools for the different disciplines: A team may choose the most beneficial method and tool – without compromise. Added value is created by

setting the results of ,any' modeling effort into a common context. So far disparate models shall be semantically integrated in certain aspects of interest. Specific model elements are mapped to abstract ones in the integration model. By exploring the results in a common context it is possible to get insight into mutual dependencies and to uncover inconsistencies.

Imagine the results of different modeling techniques can be assessed side-by-side in a common context. Which elements are conceptually the same and are comparable, therefore? How to identify the same entities in different model views? A variety of graphical notations and model element types is used in different methods. There are many conceptual similarities, though.

Based on the approach of the [Fundamental Modeling Concepts](#) (FMC), developed by Prof. Siegfried Wendt and his team in the 1970s, and considering widely used model elements in system specifications, the following abstract model element types (ObjectTypes) are proposed for SpecIF to realize the semantic integration:

- A  *Diagram* is a model diagram with a specific communication purpose, e.g. a business process, a system composition or a schematic drawing.
- An  *Actor* is a fundamental model element type representing an active entity, be it an activity, a process step, a function, a system component or a user role.
- A  *State* is a fundamental model element type representing a passive entity, be it a value, an information store, even a color or shape.
- An  *Event* is a fundamental model element type representing a time reference, a change in condition/value or more generally a synchronization primitive.
- A  *Feature* is an intentional distinguishing characteristic of a system, often a so-called 'Unique Selling Proposition'.
- A  *Requirement* is a singular documented physical and functional need that a particular design, product or process must be able to perform.
- A  *Collection* is a logical (often conceptual) group of elements or other collections. Examples for collections are groups in BPMN or boundaries on use case diagrams in UML/SysML.
- A  *Package* is used to organize elements hierarchically to achieve an easy navigation in big models or element collections. Examples for packages are the packages in UML/SysML models or folders in a file system etc.

With these few fundamental elements, model integration can be done with SpecIF. Each individual element of a specific tool or specific modeling language is mapped to these fundamental element types. This creates a semantic integration layer for any kind of engineering model data, because all elements are mapped to this common base.

The SpecIF user or a SpecIF tool can always use or display at least this abstract semantic data representation with well defined semantics, to get a basic understanding of the model data, represented by SpecIF.

A description how to apply the fundamental elements for model integration is given in more detail in the chapter *Introduction to SpecIF Model Integration*.

2.15 Model Integration Guides

Through the definition of data format (syntax), term definitions (semantics) and fundamental elements to represent engineering models (semantic integration) a big step forward is achieved in comparison with other PLM data formats. Using these definitions, a tool vendor also knows how to map more complex data structures to SpecIF.

For example an UML- or SysML-model has several options of representation in the syntax and semantics of SpecIF. To develop a common understanding and to achieve a data exchange between tools, the SpecIF standard shall define model integration guidelines describing the different scenarios how tool content has to be mapped to and from SpecIF. This is the purpose of the SpecIF model integration guides.

2.16 Versioning of this specification

This specification document consists of different parts. Together they describe all concepts behind SpecIF.

Some parts of SpecIF have their own version numbers/revisions to allow for independent development steps.

This specification describes the release of SpecIF 1.1, which is the first official release of SpecIF. It is not called 1.0, because the development of the different parts of SpecIF took some time and parts were presented earlier as version 1.0. To avoid misunderstandings and confusion, the first official release is called SpecIF 1.1!

This release of *SpecIF 1.1* consists of the following parts, described in the following chapters:

Title	Revision
SpecIF Metamodel and SpecIF JSON-schema	1.1
SpecIF class definitions and SpecIF Vocabulary	1.1
SpecIF Web API	1.1

3 SpecIF Metamodel

This section describes the SpecIF metamodel used to define the SpecIF syntax.

The description by a metamodel is done in a technology independent way following the OMG [Meta Object Facility \(MOF\) Metamodeling](#) technology and the OMG [Model Driven Architecture \(MDA\)](#) approach. The metamodel defines a platform-independent model (PIM).

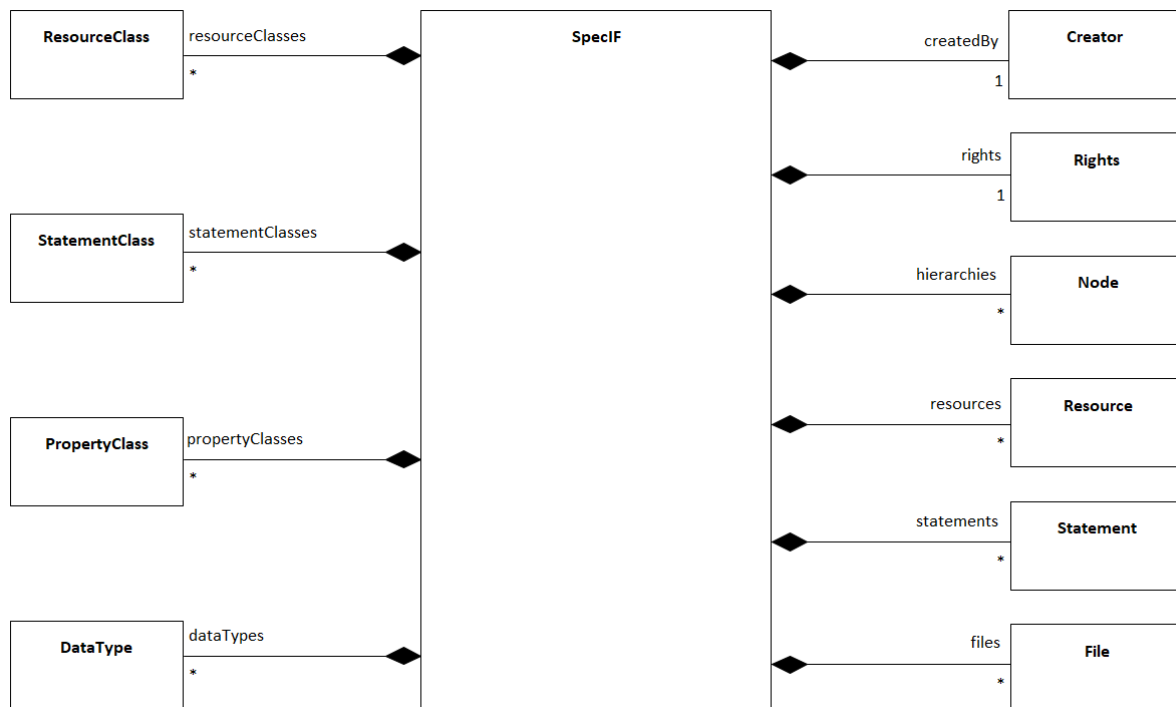
The MOF metamodeling approach is a simplified form of UML class modeling to define data structures (terms, attributes and associations) for data models. From such a platform-independent metamodel, different concrete data representations - platform-specific models (PSM) - can be derived. This can for example be a SpecIF representation using JSON, XML or a SQL database schema and so on.

You will find the realization as JSON schema, used for file- and Web API-representations in the next chapters of this specification.

Further platform-specific models for SpecIF (e.g. XML) are not yet defined and are not part of this release version of the SpecIF-specification, but may be developed and released in future releases of SpecIF.

3.1 SpecIF Repositories

The following figure shows the entry point to the SpecIF metamodel. The attributes of the classes are hidden for better readability and will be discussed in the sections below.



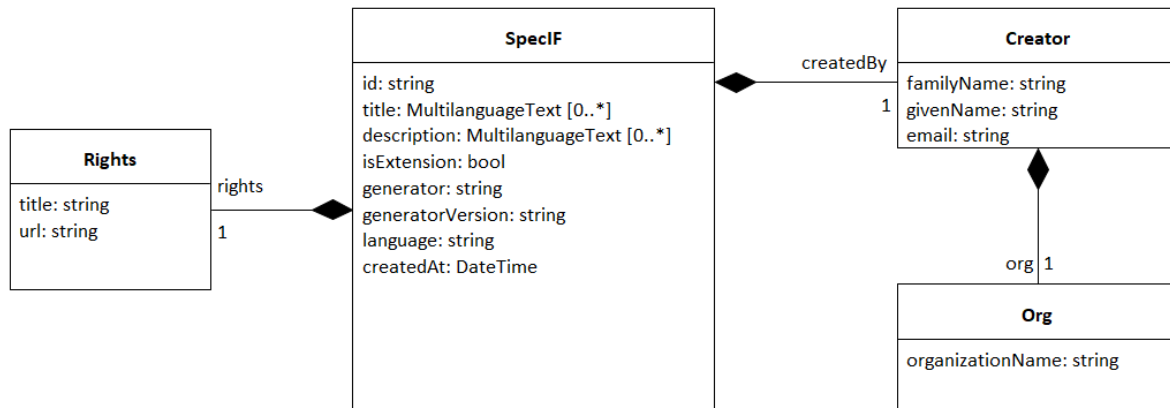
SpecIF Project Metamodel

The SpecIF class is the outermost element of a SpecIF data representation. We can call it a repository or data set. It defines the entire SpecIF data set represented in platform-specific implementation by a SpecIF file or a SpecIF persistence layer like a database or data provided by a Web API. Such a SpecIF repository instance is often identical with a project context in a development project. It can contain the entire content of PLM data created in a project context.

SpecIF allows the selected and distributed inclusion of data into a SpecIF repository. It is not required to have all data, that can be stored in a SpecIF data set, in a single repository. Therefore, you can have one SpecIF repository containing the data and class definitions, and a second, separated set containing the data content, but referencing the required data types by the unique identifiers. A SpecIF instance has a boolean attribute called *isExtension*. If this value is set to true, a tool, that is working with this data, needs further SpecIF data to get the complete definition of all required data types and data type definitions.

3.1.1 SpecIF-Repository description attributes

To express additional information about the SpecIF repository resp. the project data represented by SpecIF, the metamodel defines some attributes for describing the SpecIF metaclass.



The SpecIF repository metaclasses

3.1.1.1 SpecIF

The SpecIF metaclass has the following attributes:

- *id*: *string* - A unique identifier for the SpecIF repository resp. the project represented by the SpecIF repository/file.
- *title*: *MultilanguageText[]* - A human readable description for the entire SpecIF repository/project.
- *description*: *MultilanguageText[]* - A human readable description for documentation purposes.
- *isExtension*: *bool* - Indicates that the project is not schema-compliant on its own; by default the value is 'false'. Of course, it is expected that once extended the project is schema-compliant.
- *generator*: *string* - The generator that generates the SpecIF data.
- *generatorVersion*: *string* - The SpecIF generator tool version.
- *language*: *string* - An IETF language tag such as 'en', 'en-US', 'fr' or 'de' showing the used language of simple property values. Is superseded by a resource's, statement's or property's language value.
- *createdAt*: *DateTime* - The creation date.
- *createdBy*: *Creator* - The creator of the SpecIF data set. If specified, at least an e-mail address must be given.
- *rights*: *Rights* - Copyright and/or license information.

3.1.1.2 Creator

The *Creator* metaclass defines information about the SpecIF data creator.

It has the following attributes:

- *familyName*: *string* - The creator's family name.
- *givenName*: *string* - The creator's given name.
- *email*: *string* - The creator's e-mail.
- *org*: *Org* - The creator's organization.

3.1.1.3 Org

The *Org* metaclass describes the SpecIF creator's organization and has the following attribute:

- *organizationName*: *string* - The organization name.

3.1.1.4 Rights

The *Rights* metaclass defines a data structure to represent copyright and/or license information about the SpecIF data. It has the following attributes:

- *title*: *string* - A title for the copyright/license information (e.g. 'Apache license v2').
- *url*: *string* - A valid uniform resource locator to further copyright/license information.

3.2 Data representation and data type definitions in SpecIF

In SpecIF it is possible to define concrete data using the concept of graph data, represented by the metaclasses *Resource* (as graph nodes) and *Statement* (as graph edges). These two elements can contain *Property* elements to store a set of data elements (e.g. a title and a description). A property has a well defined (primitive) data type (e.g. string, integer etc.). This allows a tool to present, edit and validate property data using specialized editors for numbers, text or formatted text.

The elements containing the data need a specification about data structure and data types. This is done in SpecIF by defining a set of classifiers for the concrete data. These classifiers define for example what kind of properties are included in a resource element called 'Requirement'.

The SpecIF-Metamodel defines a set of classes that are responsible for these data type definitions:

- The *ResourceClass* defines the type of a *Resource*.

- The *StatementClass* defines the type of a *Statement*.
- The *PropertyClass* defines the type of a *Property*.
- The *DataType* defines the primitive data types (Integer, String, Double, Boolean, DateTime, Enumerations) used as data types for property definitions. The type string can represent formatted text or unformatted text.

3.3 Metamodel helper classes

The SpecIF metamodel defines three helper classes. They are described in this section.

3.3.1 Base Element

Some attributes in the metamodel are common for multiple metaclasses. We define this in an abstract metaclass *BaseElement*.

<i>BaseElement</i>
id: string revision: string replaces: string[] changedAt: DateTime changedBy: string

SpecIF metaclass base attributes

- *id: string* - A unique identifier for the SpecIF element.
- *revision: string* - A unique identifier for the revision of the SpecIF element.
- *replaces: string[]* - The revision IDs of the SpecIF elements replaced by this element revision. This array has a maximum length of 2 entries and can contain 0 entries (no predecessor), 1 entry (1 predecessor) or 2 entries (this element is merged from 2 predecessors).
- *changedAt: DateTime* - The date and time when the element was changed.
- *changedBy: string* - The change author.

The following metaclasses are inherited from *BaseElement*:

- *DataType*,
- *PropertyClass*,
- *ResourceClass*,
- *StatementClass*,
- *Resource*,
- *Statement*,
- *Node*,

- File.

Remark: The attributes of the base metaclass `BaseElement` are not set for each child class anymore!

3.3.2 MultilanguageText

MultilanguageText
<code>text: string</code> <code>format: TextFormat</code> <code>language: string</code>

«enumeration» TextFormat
<code>plain</code> <code>xhtml</code>

The metaclass `MultilanguageText`

To realize the concept of data storage in multiple languages, SpecIF defines a metaclass called *MultilanguageText*.

This class defines three attributes:

- *text: string* - This attribute contains the data content in a certain language.
- *format: TextFormat* - An enumeration value describing the text format. Allowed values are *plain* for plain text or *xhtml* for formatted content.
- *language: string* - An IETF language tag such as 'en', 'en-US', 'fr' or 'de' showing the used language.

Typically an array of these `MultilanguageText` objects is used to represent a value in multiple languages. Each single array value contains the same text content but in a specific language (english, german, etc.).

3.3.3 EnumeratedValue

EnumeratedValue
<code>id: string</code> <code>value: MultilanguageText[]</code>

The metaclass `EnumeratedValue`

To define enumerations in SpecIF the metaclass *EnumerationValue* is used to define the values for enumerations in data types. In SpecIF different data types can be defined as enumerations by defining a set of predefined values. This is done using the *EnumeratedValue* class.

This class defines two attributes:

- *id: string* - An identifier for the enumeration value. This ID must be unique within the collection of enumeration values for an enumeration data type definition.
- *value: MultilanguageText[]* - The enumeration value definition in multiple languages.

3.3.4 AlternativId

AlternativId
id: string revision: string project: string

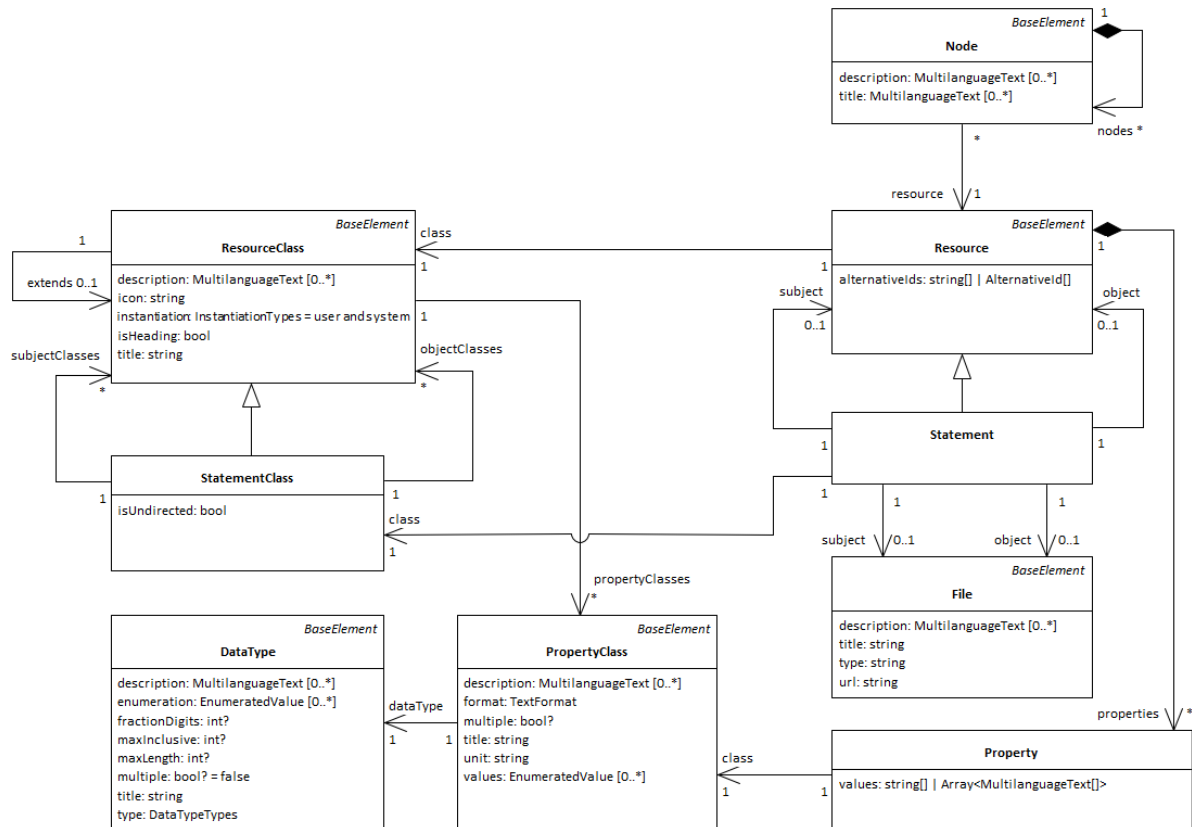
The metaclass AlternativId

AlternativId data values are used to further define identifier values in addition to the *id* attribute in SpecIF. This is helpful for data integrations where multiple tools use their own internal identifiers. So you can represent these ‘legacy’ IDs in SpecIF. *AlternativId* has the following attributes:

- *id: string* - A string with a valid identifier of an element.
- *revision: string* - A unique revision identifier.
- *project: string* - A string with a valid project identifier.

3.4 SpecIF-Metamodel details

The figure below shows the metamodel with properties and associations between elements to represent data representation, data type definitions and hierarchical structures.



SpecIF Metamodel elements with Associations

On the right you can see the classes defining the data structures for concrete data elements like *Resource*, *Statement*, *Property*, *Hierarchy* and *Node*. On the left you can see the data type defining elements like *ResourceClass*, *StatementClass*, *PropertyClass* and *DataType*.

In the following sections the semantics of all elements is described in detail.

3.4.1 DataType

<i>BaseElement</i>	<i>«enumeration»</i>
DataType description: MultilanguageText [0..*] enumeration: EnumeratedValue [0..*] fractionDigits: int? maxInclusive: int? maxLength: int? multiple: bool? = false title: string type: DataTypeTypes	DataTypeTypes xs:boolean xs:dateTime xs:anyURI xs:integer xs:double xs:string xs:duration

The metaclass DataType

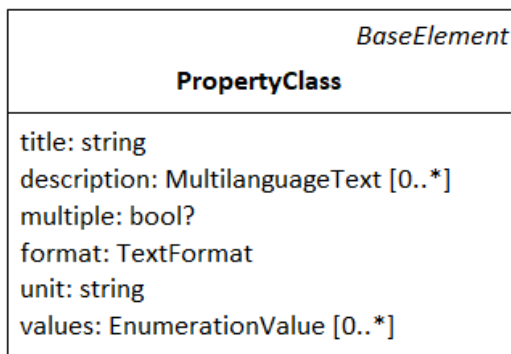
A *DataType* is used to define the base data types (primitive data types) on which all other type definitions are based. SpecIF allows the definition of primitive data types for numbers, formatted or unformatted text strings, DateTime values and enumerations. So the *DataType* metaclass defines the following attributes to satisfy these requirements:

- *title: string* - A unique name for the defined data type (e.g. 'string').
- *description: MultilanguageText[]* - A human readable description of the data type for documentation purposes.
- *type* - A formal definition of the used base type. The allowed types are taken from the XML-stylesheet-standard (<https://www.w3.org/TR/xmlschema11-1/>) and defined in the metamodel as Enumeration *DataTypeTypes* as follows:
 - xs:boolean,
 - xs:integer,
 - xs:double,
 - xs:string,
 - xs:anyURI,
 - xs:dateTime,
 - xs:duration.
- *maxInclusive* - The maximum value for a numeric data type.
- *fractionDigits* - The number of digits for floating point numbers.
- *enumeration* - A list of enumerated values.
- *multiple: bool* - This flag indicates for enumeration definitions, that a multiple selection of enumeration values should be possible.
- *maxLength: int* - The maximum length of a text value.

It is possible to restrict the possible values for a data type. Enumerated data types can be defined and used in this manner. The allowed values are defined by the attribute *enumeration* using a list of *EnumeratedValue* elements.

Because of the fact, that different data types are allowed for an enumeration definition, it is possible to define for example a set of strings as enumeration or a set of numbers etc. This might be helpful for example to define a data type for story points used in agile project management, often defined as Fibonacci numbers.

3.4.2 PropertyClass

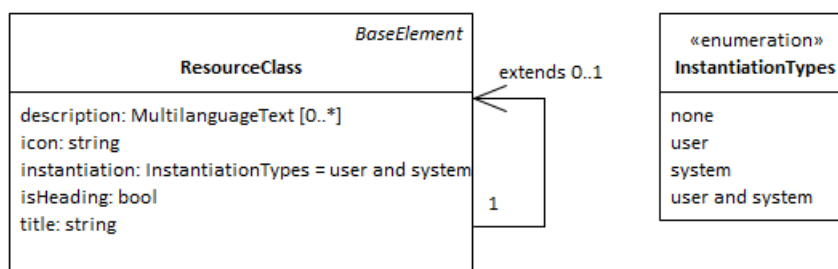


The metaclass PropertyClass

A *PropertyClass* defines the type of a SpecIF property. Properties are used to define data of resource and statement elements. The *PropertyClass* has the following attributes:

- *title: string* - A unique name for the defined PropertyClass (e.g. 'dterms:title').
- *description: MultilanguageText[]* - A human readable description of the data type for documentation purposes.
- *multiple: bool* - This flag indicates that the property value can hold multiple values (multiple enumeration values or an array of primitive data).
- *dataType* - This association references the used *DataType* for the PropertyClass.

3.4.3 ResourceClass



The metaclass ResourceClass

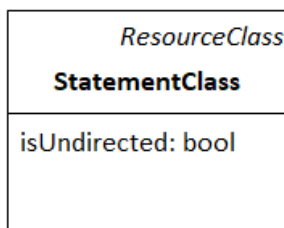
A *ResourceClass* is used to define the type of a resource element. In SpecIF the concept of inheritance is supported for data type definition. The resource class has an attribute *extends*,

where it can extend other existant resource class elements. It is possible to add new properties and reuse the existing ones from the base elements - as known from inheritance concepts in object oriented data modeling and programming.

The *ResourceClass* has the following attributes:

- *title: string* - A unique name for the defined ResourceClass (e.g. 'IREB:Requirement').
- *description: MultilanguageText[]* - A human readable description of the data type for documentation purposes.
- *icon: string* - An icon definition usable by authoring tools for resources. This can be a language code of a unicode symbol or a base64 encoded image.
- *isHeading: bool* - Indicates that the defined resource is a heading.
- *instantiation: InstantiationTypes* - Values: 'user' and/or 'system'. Indicates whether an instance of the class is created automatically, manually or both. All is allowed by default. The class is abstract and cannot be instantiated, if the value is set to 'none'.
- *extends* - Reference to a parent resource class element when inheritance is used in the data definition.
- *propertyClasses* - A list of property class references to define which properties shall be used for the defined resource type.

3.4.4 StatementClass



The metaclass StatementClass

A *StatementClass* inherits the *ResourceClass* and defines the data type definition of a SpecIF statement. Statements are the edges in a SpecIF graph data structure. A statement has two ends called *subject* and *object*.

The *StatementClass* allows the definition of possible resource types for the subject an object elements of the statement. This is done by referencing the allowed subject *ResourceClass* elements and object *ResourceClass* elements.

The *StatementClass* has the following attributes:

- *isUndirected: bool* - This flag indicates that a statement defined by this statement class has no direction. It can be used and navigated in both directions.
- *subjectClasses* - A collection of references to ResourceClass elements to define the allowed types for the statement subject.
- *objectClasses* - A collection of references to ResourceClass elements to define the allowed types for the statement object.

3.4.5 Property

Property
values: string[] Array<MultilanguageText[]>

The metaclass Property

A *Property* is an instance of a PropertyClass and is used to store a concrete data value in SpecIF. Each property in SpecIF is defined as an array of values. The attribute *isMultiple* in the PropertyClass specifies if a property value is an array or a single value.

If *isMultiple* is set to false and the property attribute values contains more than one value, only the first value shall be used and all other array elements shall be ignored!

Depending on the DataType, that is used as definition of the PropertyClass, each value of the array is either a MultilanguageText for plain or xml-textual content or a string for all other contents (e.g. numbers, enumerations etc.). For enumerations the ID of the EnumerationValue is stored as property value.

A property has the following attributes:

- *values* - The values of the property to store the property content data (multiple or single values).
- *class* - A reference to the PropertyClass element defining the property type.

3.4.6 Resource

<i>BaseElement</i>
Resource
<code>alternativeIds: string[] Alternativeld[]</code>

The metaclass Resource

A *Resource* is an instance of a *PropertyClass* and the element in SpecIF that represents a node in the graph data structure. Resources represent all kind of concrete data in PLM. This might be a requirement, a model element in UML or SysML or an electrical circuit in an E-CAD model etc.

A resource has the following attributes:

- *alternativeIds* - Alternative ID values are used to define further ID values in addition to the *id* attribute in SpecIF. This is helpful for data integrations where multiple tools have their own internal IDs. So you can represent these ‘legacy’ IDs in SpecIF.
- *class* - A reference to the *ResourceClass* element defining the resource type.
- *properties* - A collection of property elements to store property data for the resource.

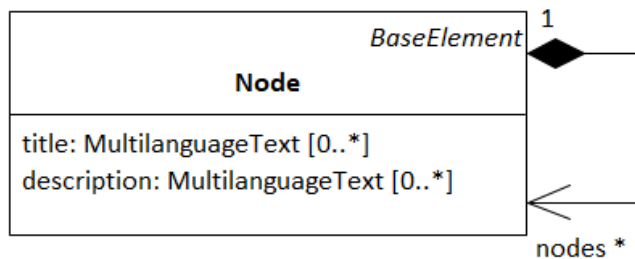
3.4.7 Statement

A *Statement* is an instance of a *StatementClass* and the element in SpecIf that defines the edge in the graph data structure. Statements allow to define predicate logic in form of *Subject - Predicate - Object*. The subject and the object are resources, statements of file elements, the predicate is always a statement.

A statement is inherited from a resource and has the following additional attributes:

- *subject* - A reference to the resource, statement or file that is used as subject for the statement.
- *object* - A reference to the resource, statement or file that is used as object for the statement.

3.4.8 Node



The metaclass Node

The metaclass *Node* allows to define hierarchical data structures (trees) in SpecIF. Typical application scenarios are hierarchical structures in textual specifications like documents and the structures in CAD and other modeling tools.

The data is not directly included inside node elements. Instead, the resources of a node are just included via reference. This allows a separation of data model and view (view concept as known e.g. from UML-tools) and the inclusion of the same resource elements into multiple hierarchy trees.

The node element have the following attributes:

- *title: MultilanguageText[]* - A title for the node. Mostly used for internal purposes, because the title normally comes from the referenced resource element properties.
- *description: MultilanguageText[]* - A human readable description for documentation purposes. Mostly used for internal purposes, because the description normally comes from the referenced resource element properties.
- *nodes* - A collection of child node elements for the node.
- *resource* - A reference to the resource element used as Node tree node data.

3.4.9 File

The last remaining element to describe is the *File* metaclass. SpecIF allows the inclusion of any kind of file into the repository. This allows for example to add files as attachments to a SpecIF specification etc. The File element allows to describe and reference files contained in a SpecIF repository.

<i>BaseElement</i>	
File	
+	title: string
+	description: MultilanguageText [0..*]
+	type: string
+	url: string

The metaclass File

The metaclass *File* has the following attributes:

- *title: string* - In case of a file, the title is the absolute or relative URL.
- *description: MultilanguageText[]* - A human readable description for documentation purposes.
- *type: string* - The file's media type (formerly MIME-type) according to <https://www.iana.org/assignments/media-types/media-types.xhtml>.
- *url: string* - An absolute or relative URL to the file. If missing, the title applies.

4 SpecIF JSON-Schema

The SpecIF JSON-schema describes the syntax of SpecIF data as a concrete instance of the SpecIF-Metamodel, like it was described earlier in this specification. The JSON-schema builds a platform-specific model (PSM), following the OMG MDA approach, realizing a JSON representation for SpecIF, based on the SpecIF metamodel. The schema follows the JSON-schema standard defined here: <https://json-schema.org/draft/2019-09/schema#>.

The schema definition file for SpecIF is available under

- <https://specif.de/v1.1/schema.json> or
- <https://json.schemastore.org/specif-1.1.json>

Remark: The names, internally used in the JSON-schema for SpecIF all get a prefix *Specif*, followed by the names defined in the metamodel to avoid naming conflicts in implementations and code, generated from the schema definition, with equal names defined by other standards (e.g. the term *Node* is used in the JavaScript world by NodeJS and also defined in the SpecIF metamodel). So the metamodel term *Node* is called *SpecifNode* in the JSON-schema definition etc.

4.1 SpecIF JSON example

The following example shows an empty SpecIF JSON-object to demonstrate the principle of data representation within SpecIF using JSON. Not all JSON-properties are set, but the SpecIF schema defines just a subset as mandatory:

```
{
  "$schema": "https://specif.de/v1.1/schema.json",
  "id": "_3555D75E_0344_4BDF_B127_2340C7F2BF9A",
  "title": [
    {
      "text": "Empty SpecIF file",
      "format": "plain",
      "language": "en"
    },
    {
      "text": "Eine leere SpecIF-Datei",
      "format": "plain",
      "language": "de"
    }
  ],
  "isExtension": false,
  "dataTypes": [],
  "resourceClasses": [],
  "statementClasses": []
}
```

```
"resources": [],  
"statements": [],  
"hierarchies": [],  
"files": []  
}
```

4.2 Definition elements and data elements

The data and class definition in SpecIF is a cascading data structure: At first a data type is defined out of a set of primitive data types. The list below shows the available primitive data types based on the data types defined in XML-schemas (XSD):

- *xs:string* - a formatted or unformatted text string,
- *xs:boolean* - a boolean value,
- *xs:integer* - a number value,
- *xs:double* - a floating point number,
- *xs:dateTime* - a date value,
- *xs:anyURI* - a uniform resource identifier (URI) including a uniform resource locator (URL),
- *xs:duration* - a duration value.

Based on these primitive data types, a SpecIF user can define *DataType* elements. These data types can be used as base for *PropertyClass*-definitions. The property class definitions can then be used to define *ResourceClass*- and *StatementClass*-definitions. Using this approach all kinds of data representation can be defined in SpecIF.

The resources, statement, properties and hierarchies instantiate the class definitions. A SpecIF-tool can automatically generate a property editor for a resource or statement element, because all necessary information is available in the class and data type definition elements.

A reference to another SpecIF element is always expressed using the *Key* -helper-element. It contains the *ID* and the *revision-ID* of the referenced element. If no *revision* is given, the element with the newest *changedAt* date shall be used.

4.3 Definition elements

4.3.1 Data types

SpecIF data types define the data types used in a property to store a PLM data value. All data types are defined out of the primitive data types listed above. All data types, except enumerations, are defined in a very similar way.

The JOSN-snippet below shows some data type definitions for non-enumeration types:

```
{
  "id": "DT-Boolean",
  "title": "Boolean",
  "description": [
    {
      "text": "The Boolean data type.",
      "format": "plain",
      "language": "en"
    }
  ],
  "revision": "1.1",
  "replaces": [],
  "type": "xs:boolean",
  "changedAt": "2016-05-26T08:59:00+02:00"
},
{
  "id": "DT-Byte",
  "title": "Byte",
  "description": [
    {
      "text": "A byte is an integer value in range between 0 and 255.",
      "format": "plain",
      "language": "en"
    }
  ],
  "revision": "1.1",
  "replaces": [],
  "type": "xs:integer",
  "minInclusive": 0,
  "maxInclusive": 255,
  "changedAt": "2016-05-26T08:59:00+02:00"
},
{
  "id": "DT-Integer",
  "title": "Integer",
  "description": [
    {
      "text": "A numerical integer value from -32768 to 32767.",
      "format": "plain",
      "language": "en"
    }
  ],
  "revision": "1.1",
  "replaces": [],
  "type": "xs:integer",
  "minInclusive": -32768,
  "maxInclusive": 32767,
  "changedAt": "2016-05-26T08:59:00+02:00"
},
{
  "id": "DT-Real",
  "title": "Real",
  "description": [
    {
      "text": "A floating point number (double).",
      "format": "plain",
      "language": "en"
    }
  ],
  "revision": "1.1",
  "replaces": [],
  "type": "xs:double",
  "minInclusive": -1.7976931348623157e+308,
  "maxInclusive": 1.7976931348623157e+308,
  "changedAt": "2016-05-26T08:59:00+02:00"
}
```




```
"revision": "1.1",
"replaces": [],
"type": "xs:double",
"changedAt": "2021-02-14T08:59:00+02:00"
},
{
  "id": "DT-Decimal2",
  "title": "Real with 2 Decimals",
  "description": [
    {
      "text": "A floating point number (double) with two fraction digits.",
      "format": "plain",
      "language": "en"
    }
  ],
  "revision": "1.1",
  "replaces": [],
  "type": "xs:double",
  "fractionDigits": 2,
  "changedAt": "2021-02-14T08:59:00+02:00"
},
{
  "id": "DT-DateTime",
  "title": "Date or Timestamp",
  "description": [
    {
      "text": "Date or Timestamp in ISO-Format",
      "format": "plain",
      "language": "en"
    }
  ],
  "revision": "1.1",
  "replaces": [],
  "type": "xs:dateTime",
  "changedAt": "2016-05-26T08:59:00+02:00"
},
{
  "id": "DT-ShortString",
  "title": "String[256]",
  "description": [
    {
      "text": "String with max. length 256",
      "format": "plain",
      "language": "en"
    }
  ],
  "revision": "1.1",
  "replaces": [],
  "type": "xs:string",
  "maxLength": 256,
  "changedAt": "2016-05-26T08:59:00+02:00"
},
{
  "id": "DT-Text",
  "title": "Plain or formatted Text",
  "description": [
    {
      "text": "An account of the resource (source: http://dublincore.org/documents/dcmi-terms/). Descriptive text represented in plain or rich text using XHTML. SHOULD include only content that is valid and suitable inside an XHTML <div> element (source: http://open-services.net/).",
      "format": "plain",
```



```
        "language": "en"
      },
    ],
    "revision": "1.1",
    "replaces": [],
    "type": "xs:string",
    "changedAt": "2021-02-23T08:59:00+02:00"
  },
  {
    "id": "DT-URL",
    "title": "URL",
    "description": [
      {
        "text": "A uniform resource locator.",
        "format": "plain",
        "language": "en"
      }
    ],
    "revision": "1.1",
    "replaces": [],
    "type": "xs:string",
    "maxLength": 1024,
    "changedAt": "2016-05-26T08:59:00+02:00"
  },
  {
    "id": "DT-EmailAddress",
    "title": "E-mail",
    "description": [
      {
        "text": "Data type to represent an E-mail address.",
        "format": "plain",
        "language": "en"
      }
    ],
    "revision": "1.1",
    "replaces": [],
    "type": "xs:string",
    "maxLength": 256,
    "changedAt": "2016-05-26T08:59:00+02:00"
  }
}
```

4.3.2 Enumerations

Any data type except '*xs:boolean*' may define a set of enumerated values using the attribute *enumeration*. If defined, only those discrete values are eligible. For example, - a data type to be used for priority can be defined as '*xs:string*' with the enumerated values [*'high'*, *'medium'*, *'low'*]. - a data type to be used for an effort in planning poker can be defined as '*xs:integer*' with the enumerated values [*0*, *1*, *2*, *3*, *5*, *8*, *13*, *21*, *34*, *55*, *89*].

SpecIF supports enumeration with multiple selection, so more than one value of an enumeration can be selected in a SpecIF property. If the JSON-attribute *multiple* is set to *true*, multiple selection is allowed.

The following data type definitions show an example for an enumeration data type definition with SpecIF using the *xs:string* data type:

```
{
  "id": "DT-LifeCycleStatus",
  "title": "SpecIF:LifeCycleStatus",
  "description": [
    {
      "text": "Enumerated values for status"
    }
  ],
  "revision": "1.1",
  "replaces": [],
  "type": "xs:string",
  "enumeration": [
    {
      "id": "V-Status-0",
      "value": [
        {
          "text": "SpecIF:LifecycleStatusDeprecated"
        }
      ]
    },
    {
      "id": "V-Status-1",
      "value": [
        {
          "text": "SpecIF:LifecycleStatusRejected"
        }
      ]
    },
    {
      "id": "V-Status-2",
      "value": [
        {
          "text": "SpecIF:LifecycleStatusInitial"
        }
      ]
    },
    {
      "id": "V-Status-3",
      "value": [
        {
          "text": "SpecIF:LifecycleStatusDrafted"
        }
      ]
    },
    {
      "id": "V-Status-4",
      "value": [
        {
          "text": "SpecIF:LifecycleStatusSubmitted"
        }
      ]
    },
    {
      "id": "V-Status-5",
      "value": [
        {

```



```
        "text": "SpecIF:LifecycleStatusApproved"
      }
    ],
    {
      "id": "V-Status-8",
      "value": [
        {
          "text": "SpecIF:LifecycleStatusReady"
        }
      ]
    },
    {
      "id": "V-Status-6",
      "value": [
        {
          "text": "SpecIF:LifecycleStatusDone"
        }
      ]
    },
    {
      "id": "V-Status-9",
      "value": [
        {
          "text": "SpecIF:LifecycleStatusValidated"
        }
      ]
    },
    {
      "id": "V-Status-7",
      "value": [
        {
          "text": "SpecIF:LifecycleStatusReleased"
        }
      ]
    },
    {
      "id": "V-Status-10",
      "value": [
        {
          "text": "SpecIF:LifecycleStatusWithdrawn"
        }
      ]
    }
  ],
  "multiple": false,
  "changedAt": "2021-02-21T08:59:00+02:00"
}
```

4.3.3 Property classes

Property classes define the type and kind of a property. The property is the SpecIF element containing the values of the PLM data. Resource and statement elements contain an array of defined property instances to store the PLM values.

The property class definitions are very important, because here the name resp. title of a property is defined. Properties are in principle key/value pairs representing data. The *title* JSON-property of the property class defines the term used for the property.

The SpecIF standard also contains a standardized set of data type and class definitions. The properties are one integral part of the standardized SpecIF data format vocabulary (syntax) and the standardization allows the data exchange between different tools without manual data mapping.

An example of property class definitions is given below. If no format attribute is explicitly set, plain is used as default.

```
{
  "id": "PC-Name",
  "title": "dcterms:title",
  "description": [
    {
      "text": "<p>A name given to the resource. <small><i>source: <a href=\"http://dublincore.org/documents/dcmi-terms/\">DCMI</a></i></small></p><p>Title (reference: Dublin Core) of the resource represented as rich text in XHTML content. SHOULD include only content that is valid inside an XHTML <span> element. <small><i>source: <a href=\"http://open-services.net/\">OSLC</a></i></small></p>",
      "format": "xhtml",
      "language": "en"
    }
  ],
  "revision": "1.1",
  "replaces": [],
  "dataType": {
    "id": "DT-ShortString",
    "revision": "1.1"
  },
  "changedAt": "2016-05-26T08:59:00+02:00"
},
{
  "id": "PC-Description",
  "title": "dcterms:description",
  "description": [
    {
      "text": "<p>An account of the resource. <small><i>source: <a href=\"http://dublincore.org/documents/dcmi-terms/\">DCMI</a></i></small></p><p>Descriptive text (reference: Dublin Core) about resource represented as rich text in XHTML content. SHOULD include only content that is valid and suitable inside an XHTML <div> element. <small><i>source: <a href=\"http://open-services.net/\">OSLC</a></i></small></p>",
      "format": "xhtml",
      "language": "en"
    }
  ],
}
```

```

    "revision": "1.1",
    "replaces": [],
    "format": "xhtml",
    "dataType": {
      "id": "DT-Text",
      "revision": "1.1"
    },
    "changedAt": "2016-05-26T08:59:00+02:00"
  }
}

```

In the example you can see that as title for the property with the ID *PC-Name* the term *dterms:title* is used. So this property class defines a property where the name of an PLM data element can be stored.

4.3.4 Resource Classes

Resource classes define the resource types used to store PLM data. A resource class contains, similar to a property class, a definition for a title value, defining the syntax. It also defines a list of key elements pointing to the property classes to define which properties are available to store data values in a resource, based on that resource class (JSON-property *propertyClasses*).

An example resource class definition is shown below. This example shows the resource class defining the data structure to store a requirement element following the IREB recommendation.

```

{
  "id": "RC-Requirement",
  "title": "IREB:Requirement",
  "description": [
    {
      "text": "A 'Requirement' is a singular documented physical and functional need
that a particular design, product or process must be able to perform."
    }
  ],
  "revision": "1.1",
  "replaces": [],
  "icon": "&#8623;",
  "isHeading": false,
  "instantiation": [
    "user"
  ],
  "propertyClasses": [
    {
      "id": "PC-VisibleId",
      "revision": "1.1"
    },
    {
      "id": "PC-Name",
      "revision": "1.1"
    },
    {
      "id": "PC-Description",
      "revision": "1.1"
    }
  ],
}

```



```
{
  "id": "PC-RequirementType",
  "revision": "1.1"
},
{
  "id": "PC-Priority",
  "revision": "1.1"
},
{
  "id": "PC-LifeCycleStatus",
  "revision": "1.1"
},
{
  "id": "PC-Perspective",
  "revision": "1.1"
},
{
  "id": "PC-Discipline",
  "revision": "1.1"
}
],
"changedAt": "2021-02-22T08:59:00+02:00"
}
```

4.3.5 Statement Classes

Statement classes define the statements (links between SpecIF data elements - resources, statements and files). Statements start at a subject data element and end on an object data element. It is possible to restrict the subjects and objects where a statement can be used by defining entries in the *subjectClasses* and *objectClasses* JSON-properties of the statement class definition. If there are no entries, any element can be used as the statement's subject or object.

The following example of a statement class definition defines a statement that can be used between requirement resource elements. It defines the *IREB:refines* relationship. Also known as *deriveReq* dependency in SysML:

```
{
  "id": "SC-refines",
  "title": "IREB:refines",
  "description": [
    {
      "text": "The subject requirement refines the object requirement."
    }
  ],
  "revision": "1.1",
  "replaces": [],
  "instantiation": [
    "user"
  ],
  "subjectClasses": [
    {
      "id": "RC-Requirement",
      "revision": "1.1"
    }
  ]
}
```

```

    ],
    "objectClasses": [
      {
        "id": "RC-Requirement",
        "revision": "1.1"
      }
    ],
    "changedAt": "2016-05-26T08:59:00+02:00"
  }

```

4.4 Data elements

The SpecIF data elements resource and statement represent concrete PLM data. All data elements shall have a reference to the definition class element (resource class or statement class) using the *class* JSON-property element.

SpecIF-properties defined for a resource or statement are stored inside a resource or statement element containing the data value of the represented PLM data.

4.4.1 Resources

Resources are the nodes in a SpecIF data set graph. They contain the concrete data values. All elements, that are no relations resp. connectors between elements are represented as resources in SpecIF.

The following example shows a resource representing a requirement element. In the description property value the attributes for format and language are not explicitly set. So the default values are used: English as default language and the format defined in the *PropertyClass* of the property.

The property values representing textual content are using the *MultilanguageText* data structure. The status value, defined as enumeration, uses a string with the EnumerationValue identifier (ID).

```

{
  "id": "_73392B3D_CF8F_4ac0_BC77_E6A2C9415EF4",
  "revision": "F16C40BE-DFC4-46BB-85C4-FDF9433F8E73",
  "replaces": [],
  "class": {
    "id": "RC-Requirement",
    "revision": "1.1"
  },
  "properties": [
    {
      "values": [
        {

```




```
        "text" : "Login",
        "format" : "plain",
        "language" : "en"
      },
      {
        "text" : "Benutzeranmeldung",
        "format" : "plain",
        "language" : "de"
      }
    ],
    "class": {
      "id": "PC-Name",
      "revision": "1.1"
    }
  },
  {
    "values": [
      [
        {
          "text" : "The system shall provide the user with the ability to log
in."
        },
        {
          "text" : "Das System muss dem Benutzer die Möglichkeit bieten sich
anzumelden.",
          "language" : "de"
        }
      ]
    ],
    "class": {
      "id": "PC-Description",
      "revision": "1.1"
    }
  },
  {
    "values": [
      "V-Status-5"
    ],
    "class": {
      "id": "PC-LifecycleStatus",
      "revision": "1.1"
    }
  }
],
"changedAt": "2021-03-07T11:16:21",
"changedBy": "oa"
}
```

4.4.2 Statements

Statements are used in SpecIF to define relationships between concrete SpecIF data elements. Normally they express relationships between two resource elements. In some special cases they can also express relationships between two statements or a resource and a statement. This depends on what the given key for *subject* and *object* references.

Statements can have a list of properties to store additional data.

The following example shows a statement example. This statement has no properties defined.

```
{
  "id": "_2186cb8d_390f_427c_9df3_a9756763b6ed",
  "revision": "820BFEE3-5808-4395-A0C2-F11E27FAFE59",
  "replaces": [],
  "class": {
    "id": "SC-contains",
    "revision": "1.1"
  },
  "subject": {
    "id": "_7BBB98BF_6966_46cd_A2B4_677B15CEC761",
    "revision": "2B750A3F-23B5-4ACB-9D70-FB09D9C2604F"
  },
  "object": {
    "id": "_73392B3D_CF8F_4ac0_BC77_E6A2C9415EF4",
    "revision": "E6F0C4F3-6939-4A82-B4D9-2E48B702B8A6"
  },
  "properties": [],
  "changedAt": "2021-03-29T15:13:54.0813277+02:00",
  "changedBy": "oa"
}
```

4.5 Hierarchies and Nodes

The *hierarchies* array in the SpecIF data set contains a collection of *node* SpecIF elements. It is possible to create a hierarchical view to a selected set of resource elements, using node elements. The *Node* is defined as a recursive data structure. Each node includes a reference to a selected resource (JSON-property *resource*) and the node can contain a collection of child nodes in the JSON-property *nodes*. This allows using the node to define a hierarchical view to SpecIF resource elements.

A typical application scenario is the representation of a chapter structure from textual requirements specifications.

The following example shows the application of nodes to create a hierarchy:

```
"hierarchies": [
  {
    "id": "_16d04111_9db8_4e4c_8223_7ebd6ec2abac",
    "revision": "60BF9E2F-D684-481B-B7A3-4C401DBE7762",
    "replaces": [],
    "title": [],
    "description": [
      {
        "text": "Requirement specification"
      }
    ],
    "resource": {
      "id": "_5D647B1D_D622_4a45_90EB_5FC6ECCD405C",
      "revision": "F583802E-FE82-4D10-BCBF-1B387C04A84C"
    },
    "nodes": [
```

```
{
  "id": "_5a48b367_c1c1_4184_8931_e5bb794d3fd5",
  "revision": "F583802E-FE82-4D10-BCBF-1B387C04A84C",
  "replaces": [],
  "title": [],
  "description": [
    {
      "text" : "Element: Introduction"
    }
  ],
  "resource": {
    "id": "_7BBB98BF_6966_46cd_A2B4_677B15CEC761",
    "revision": "0D737A07-8DFE-4502-958A-04BBF1B0B16F"
  },
  "nodes": [
    {
      "id": "_81b06448_c844_4452_a3ef_8a2171c597c1",
      "revision": "00875EE1-7491-4AD9-92D1-27B377FCFDCD",
      "replaces": [],
      "title": [],
      "description": [
        {
          "text" : "Element: Login"
        }
      ],
      "resource": {
        "id": "_73392B3D_CF8F_4ac0_BC77_E6A2C9415EF4",
        "revision": "F16C40BE-DFC4-46BB-85C4-FDF9433F8E73"
      },
      "nodes": [],
      "changedAt": "2019-05-31T15:13:54.0813277+02:00"
    }
  ],
  "changedAt": "2019-05-31T15:13:54.0303315+02:00"
}
],
"changedAt": "2019-05-31T15:13:53.8253311+02:00"
}
```

4.6 Files

SpecIF allows - beside the usage of JSON - the storage of native files as additional data. For that purpose, the SpecIF schema defines a data structure to describe and manage these files. The data structure contains an URL value to define the file's access path. The URL can be defined absolute or relative to the SpecIF data, where the file description is included. If the URL is missing, the title applies as access path.

```
{
  "id": "_9DCD9463_74E4_49B8_8C63_0E0877CDD47E",
  "revision": "95C00759-5CB5-49CF-B2AE-174B2C938730",
  "replaces": [],
  "title": "SystemArchitecture.png",
  "description": [
    {
      "text": "The logical system architecture as PNG image."
    }
  ]
}
```



```
],  
  "url": "./files_and_images/SystemArchitecture.png",  
  "type": "image/png",  
  "changedAt": "2021-03-30T13:13:53.8253311+02:00",  
  "changedBy": "oa"  
}
```

5 SpecIF Diagram Exchange using SVG

Diagram exchange is an essential concept of SpecIF to achieve the goal of data interchange in the entire PLM life-cycle where different tools are used that create data containing diagram drawings. To achieve this the Scalable Vector Graphics format (SVG) is used to define diagram information. SVG is a standardized format, defined by the W3C to represent (2D) graphical information as vector graphics using an XML-based format.

The SpecIF standard defines some additional XML-tags and attributes to include semantic diagram information into SVGs as SVG metadata. For this purpose SpecIF (re-)uses some tags defined in the OMG standard for Diagram Definition (DD) (OMG No.: formal/2015-06-01).

Therefore, the resulting SVG data consists of the graphical diagram information, represented as SVG. Such a SVG can be shown with all SVG viewers (e.g. a web browser), so that a user can easily view the diagram. Beside the SVG information, some SpecIF-specific meta-information is added to the SVG data. With this information, a semantical diagram exchange (export/import) between different modeling tools implementing SpecIF is possible. Typical meta-information are coordinates of the diagram nodes and edges and references to the SpecIF-resources and -statements shown in the diagram.

5.1 XML namespaces in SVG

The concept of XML-namespaces is used to differentiate the tags defined by SpecIF, SVG and OMG Diagram Definition . The following namespaces are relevant for SpecIF-SVG diagram exchange, defined by SpecIF and re-used from OMG diagram interchange (di) and diagram common (dc) standards.

- `xmlns:specif="https://specif.de/v1.1/schema-DI"`
- `xmlns:di="http://www.omg.org/spec/DD/20100524/DI"` - [OMG Diagram Interchange v1.0](http://www.omg.org/spec/DD/20100524/DI)
- `xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"` - [OMG Diagram Definition v1.1](http://www.omg.org/spec/DD/20100524/DC)

The namespaces and the used SVG version must be declared in the outermost tag (agname: `svg`) of the SVG XML-structure. Example:

```
<svg xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
      xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
      xmlns="http://www.w3.org/2000/svg"
      xmlns:specif="https://specif.de/v1.1/schema-DI"
      version="1.1"
      ...
>
```

The tags defined by SpecIF are defined by a XML-schema definition with the namespace URI `https://specif.de/v1.1/schema-DI`. Tags reused from the OMG diagram exchange specifications are defined in the other two namespaces. Typically SVG 1.1 builds the standard namespace, so each XML-tag without an explicit namespace setting is a SVG-tag.

5.2 Embedding SpecIF-SVG data into resource elements

A diagram is typically embedded in a resource element representing a diagram. For this purpose, SpecIF defines the resource class *SpecIF:Diagram*. A resource of the class *SpecIF:Diagram* has a property with the property class title named *SpecIF:Diagram*. The property value is able to store formatted XHTML-data.

There are two possibilities to include a diagram in a SpecIF data set:

5. Embedd the SVG or a binary graphic data format directly into the XHTML as XML resp. base64-encoded image data.
6. Store the diagram data in a separate image file (e.g. `diagram.svg`) and include an image-reference (*img*-tag) to the XHTML property value.

Using SVG with metadata is preferred to embedded diagram information in SpecIF. Binary graphic formats are supported as well, but you loose the possibility to relate metadata and only graphical information is exchanged.

5.3 Coordinate system

The SpecIF-conform SVG files follow the definitions of the SVG and the OMG Diagram Definition standards. The x-axis is horizontal and its coordinate values increase to the right. Negative coordinates are allowed. Similarly, the y-axis is vertical and its coordinate values increase to the bottom. Negative coordinates are allowed.

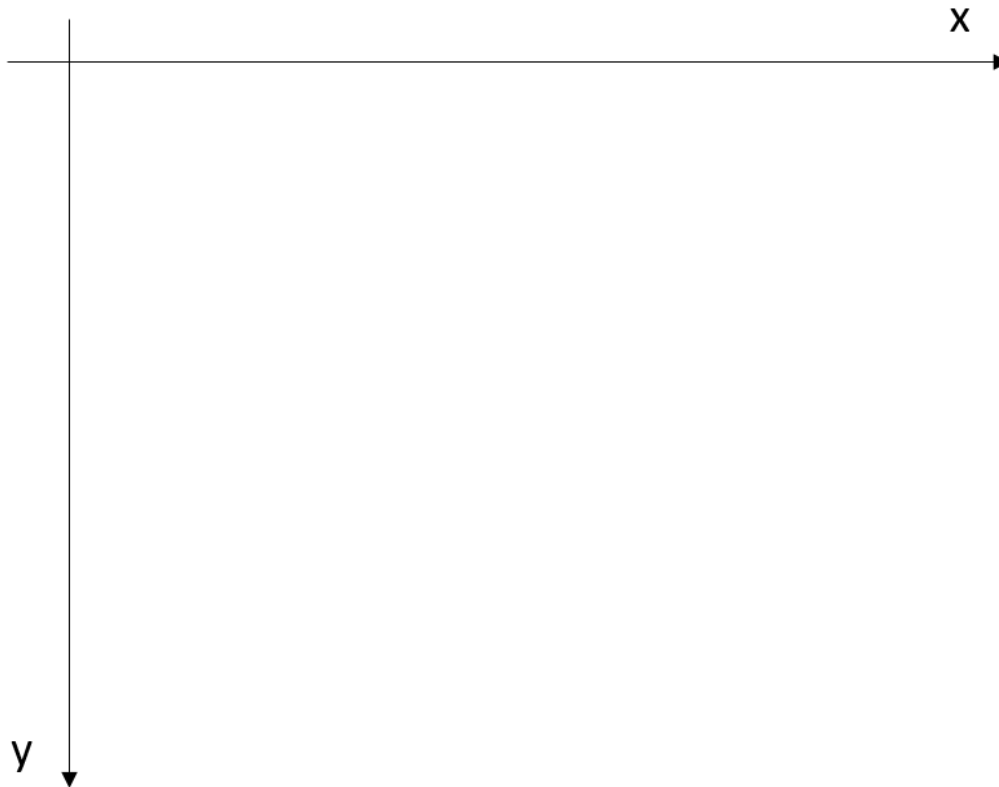


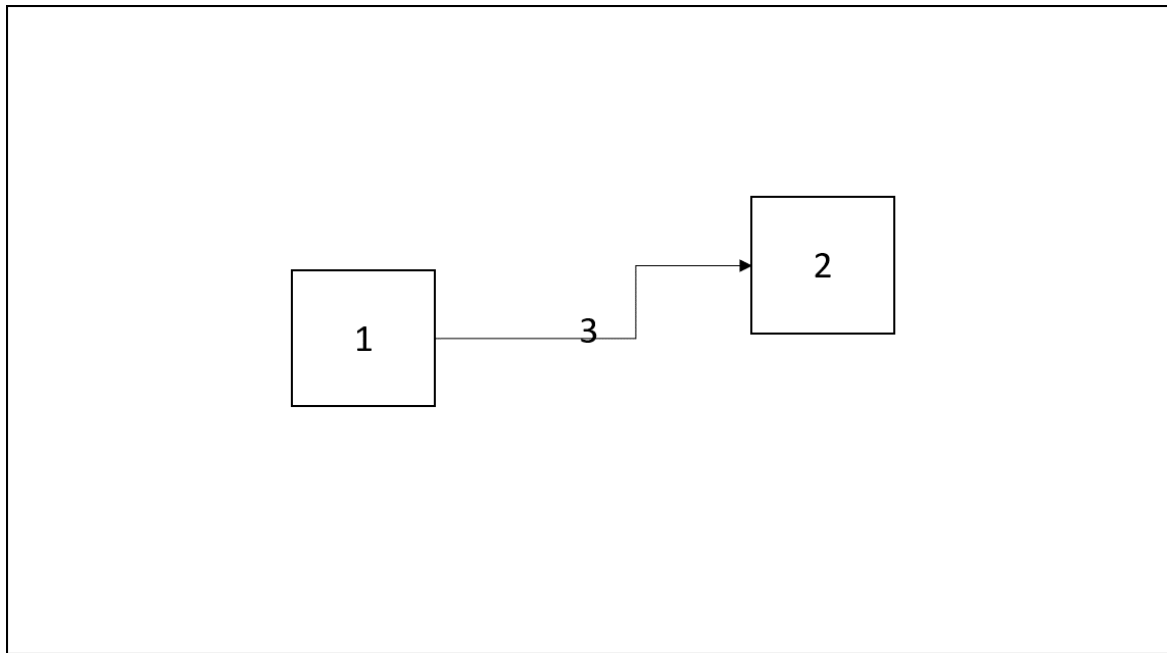
Diagram coordinate system

5.4 SVG structure and grouping

To structure SVG data in a hierarchical way, the SVG standard defines the concept of groups. A group can contain further groups as child elements as well as graphical elements like circles, rectangles or lines.

A SVG diagram that follows the SpecIF standard shall use the following structure using SVG groups:

- One SVG group represents the entire diagram as outermost element.
- Inside, a list of child groups define the graphical elements visualizing resources and statements. Therefore a SVG group is defined for each resource and statement visible on the graph.



Principle of the SpecIF SVG grouping

The image above shows this principle. The big box around is the group of the entire diagram. Inside we have three children in form of SVG groups for the three contained elements: Two boxes, named 1 and 2, and one connector, named 3.

In the SVG we will always use the following group structure for SpecIF diagram interchange example. Inside each group graphical SVG standard elements can be included to define the diagram as a visible graphic. Beside the graphical definitions each group should contain a set of semantic diagram exchange information, included inside the SVG *metadata* tags.

```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
    xmlns:specif="https://specif.de/schema/v1.0/DI"
    xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
    width="1322" height="1134" version="1.1"
    xmlns="http://www.w3.org/2000/svg">
  <g class="specif-diagram">
    <metadata>
      ... <!-- metadata for semantically diagram interchange -->
    </metadata>
    <g class="specif-resource-diagram-element" >
      <metadata>
        ... <!-- metadata for semantically diagram element interchange -->
      </metadata>
      ... <!-- SVG definition of the resource -->
    </g>
    ... <!-- more diagram elements -->
    <g class="specif-statement-diagram-element">
      <metadata>
        ... <!-- metadata for semantically diagram connector interchange -->
      </metadata>
    </g>
  </g>
</svg>
```



```
</metadata>
... <!-- SVG definition of the statement -->
</g>
... <!-- more diagram connectors -->
</g>
</svg>
```

The sequence of the group elements determine the drawing order. So the top-most element shall be defined within the last group-child.

In this example the drawing is done in this sequence: diagram, element #1, element #2, connector #3.

5.5 SpecIF diagram exchange metadata

To enable navigation from SVG to resource and statement data and to include some additional semantic information to SVG, the SpecIF standard defines tags to be included in the metadata of the SVG group elements (<g>)

For declaring groups representing resources, statements or the entire diagram, the `class`-attribute of the group tag is set to the following values:

- `specif-diagram` for the root-group, representing the entire diagram,
- `specif-resource-diagram-element` for a group defining the visualization of a SpecIF resource - child of `specif-diagram` group,
- `specif-statement-diagram-element` for a group defining the visualization of a SpecIF statement - child of `specif-diagram` group.

5.5.1 Referencing model elements

In the SVG-group defining the graphical representation of a resource, a tag called `resourceDiagramElement` is used to define a reference to the resource, the model element. The tag has an attribute to reference the ID of the resource and an attribute to reference the revision of the resource. The revision reference attribute may be missing or empty. In that case the newest revision of the resource is referenced.

The tags to define a reference to a SpecIF resource or statement are called:

- `specif:resourceReference` for references to resources,
- `specif:statementReference` for references to statements,
- `specif:sourceResourceReference` for a reference to a resource used as a connector source,

- `specif:targetResourceReference` for a reference to a resource used as a connector target.

5.6 The tags `specif:shape` and `specif:edge`

All elements in modeling tools are typically graph data, either a node or an edge. The nodes can have different kinds of shapes; the edges can have different endpoints.

To represent these elements semantically in SpecIF as SVG metadata the tags `specif:shape` for nodes and `specif:edge` for connectors are defined. Inside these metadata elements, the semantic information in form of references to SpecIF data elements, graphical coordinates and drawing information is included.

5.6.1 Diagram metadata

To represent the metadata for the entire diagram, the following structure is used:

```
<g class="specif-diagram">
  <metadata>
    <specif:shape>
      <specif:resourceReference id="_BF30DD09_E13E_451b_B210_786F93A74936"
                             revision="87accec461201406dc2b54e25f4a1f436a6763d4" />
    <dc:Bounds x="0" y="0" width="537" height="1134" />
  </specif:shape>
</metadata>
... <!-- groups specifying the elements and connectors, visible on the diagram -->
</g>
```

- The group is tagged with the class attribute *specif-diagram*,
- The metadata of the group contains a *specif:shape* child tag,
- Inside the shape tag a *specif:resourceReference* points to the SpecIF data element, that represents the diagram,
- The diagram coordinates, width and height are expressed using a *dc:Bounds*-tag.

5.6.2 Element metadata

To represent the metadata for a diagram element (node), the following structure is used:

```
<g class="specif-resource-diagram-element">
  <metadata>
    <specif:shape>
      <specif:resourceReference id="_65624FBC_FE08_4dad_971E_5D26EF996570"
                             revision="5d3656420cdeb3e4a88dd9927236dbdc23a9b946" />
    <dc:Bounds x="100" y="210" width="100" height="80" />
  </specif:shape>
</metadata>
... <!-- graphical SVG data -->
</g>
```

- The group defining the element is tagged with the class attribute *specif-resource-diagram-element*,
- The metadata of the group contains a *specif:shape* child tag,
- Inside the shape tag a *specif:resourceReference* points to the SpecIF data element, that represents the resource visualized on the diagram,
- The element coordinates, width and height are expressed using a *dc:Bounds*-tag.

5.6.3 Connector metadata

To represent the metadata for a diagram connector (edge), the following structure is used:

For a statement the following metadata structure is used, defining a SpecIF-specific tag called *statementDiagramElement*:

```
<g class="specif-statement-diagram-element">
  <metadata>
    <specif:edge direction="unidirectional"
      layoutStyle="rounded">
      <specif:statementReferences>
        <specif:statementReference id="_AE57392_ABD3_451b_B210_786F93A74936"
          revision="87acec461201406dc2b54e25f4a1f436a6763d
41">
          <specif:resourceReference id="_BF30DD09_E13E_451b_B210_786F93A74936"
            revision="87acec461201406dc2b54e25f4a1f436a6763d4
1" />
        </specif:statementReferences>
        <specif:sourceResourceReference id="_E64C5208_A8DF_45a5_B022_85C853722664"
          revision="5d3656420cdeb3e4a88dd9927236dbdc23a9b
946" />
        <specif:targetResourceReference id="_3EFC0279_A6F2_48c8_9F48_13C05288DBA6"
          revision="b6d540498c1d3720addf7714545543b3835cf
c96" />
        <di:waypoints>
          <di:Waypoint x="238" y="105" />
          <di:Waypoint x="407" y="105" />
        </di:waypoints>
      </specif:edge>
    </metadata>
    ... <!-- graphical SVG data -->
  </g>
```

- The group defining the connection is tagged with the class attribute *specif-statement-diagram-element*,
- The metadata of the group contains a *specif:edge* child tag.
- Inside the edge tag the following child tags can be present:
 - A *specif:statementReferences*-tag containing a set of *statementReference*- and/or *resourceReference*-tags. A connector on a diagram can represent, in some special cases, more than on SpecIF data element, so it is possible to define a set of references here.
 - A *specif:sourceResourceReference*-tag referencing the connector source SpecIF resource data element.

- A *specif:targetResourceReference*-tag referencing the connector target SpecIF resource data element.
- A diagram interchange waypoint collection to define the graphical coordinates of the visualized connector. It contains always at least the start and end coordinates and optional some supporting points in between.

The *specif:edge*-tag can have two optional attributes: *direction* and *layoutStyle*.

The attribute *direction* defines the connector direction. Allowed values are:

- **unidirectional** - The connector is directed from source to target.
- **bidirectional** - The connector is bi-directional.
- If the attribute is missing (not set), the connector has no direction.

The attribute *layoutStyle* specifies the layout style of the connector. Allowed values are:

- **bezier** The connector is drawn as a Bezier curve,
- **rounded** The corners of the connector in the waypoints in between start and end are connected rounded,
- If the attribute is missing (not set), the connector is drawn as a path of direct lines using the coordinates defined by the waypoint tags.

6 SpecIF Web API

The SpecIF Web API defines a set of REST endpoints for standardized access to SpecIF data available via web-services. It is possible to use such a web-service as part of a microservice software architecture.

This specification describes version 1.1 of the SpecIF Web API, which is the first official released version.

Swagger resp. OpenAPI is used to define the REST endpoints for SpecIF data access. You can find the formal API definition on GitHub under <https://github.com/GfSE/SpecIF-OpenAPI>.

6.1 Structure of the SpecIF Web API

6.1.1 CRUD operations

The SpecIF-WebAPI is designed to provide a set of create, read, update and delete (CRUD) operations for all endpoints. This makes it easy for the API users to understand the structure of the API.

6.1.2 API versioning

The SpecIF Web API uses API versioning to allow parallel run of different API versions. This allows access to different version endpoints at the same time. The API version is included in the URL of the SpecIF Web API:

```
/specif/v1.1/<endpointName>
```

The API version number defines a release version for the API itself. It does not mean that a SpecIF Web API version 1.1 is strictly assigned to a SpecIF schema 1.1. You can find a table listing the version dependencies for the different parts of SpecIF in the concepts chapter of this specification.

6.1.3 Authentication and Authorization

The SpecIF Web API should implement authorization and authentication technology to restrict the access for some endpoints to realize know-how protection or restrict the access for changing common used data like data type definition endpoints.

6.1.4 API Key Authentication

A SpecIF Web API shall provide an [API Key-based authentication](#) for API users. It is not part of the standard to define how these API keys are managed or generated. This is a task for the API developer.

The API key shall be sent inside the HTTP request header, for example:

```
X-API-KEY abcdef12345
```

API key-based authentication is only considered secure if used together with other security mechanisms such as HTTPS/SSL. It is required for all SpecIF Web APIs to use the secure HTTPS/SSL protocol. Using a non-secure HTTP protocol will lead to an error response.

6.1.5 Role-based authorization

To restrict access to some endpoints of a SpecIF Web API, role-based authorization should be implemented, using the following roles. The roles are assigned to a registered user per project, except for ‘Anybody’, which is only assigned to projects. Without role assignment, a user cannot see a project.

Role name	Role description
Anybody	Grants permission to read all instances (i.e. resources, statements, files and hierarchies) to anybody without authentication. Makes projects publicly available. The data type definition endpoints are readable to anybody if no know-how protection is necessary.
Reader	A user is granted read privilege for the project’s instances. This is the standard role with the lowest access rights. The user role is automatically applied when a user is successfully authenticated as a SpecIF Web API user.
Editor	An Editor is granted create, read, update and delete privilege for the project’s instances.
Manager	A Manager is granted create, read, update as well as delete privilege for the project’s classes and instances. In addition, projects can be created and deleted.
Administrator	An Administrator has unrestricted access to all endpoints of a SpecIF Web API.

The endpoint specification below describes if an endpoint requires further rights above the User role.

If a SpecIF Web API provides just data- and class-type definitions with public character, these endpoints are open for Anybody. If some of the data type and class type definitions have restricted usage (e.g. internal know-how protection) at least the reader role should be applied.

6.1.6 Error handling

The SpecIF Web API uses HTTP status codes for error handling. The following tables define the standard error handling behavior of the API for application endpoints. In addition, a server may send some further error messages (e.g. code 500 - Internal Server Error etc.) in exceptional situations.

6.1.6.1 Success return codes

Code	Message	Description
200	OK	The request was successful and a result was provided if necessary.
201	Created	Used for POST requests: The item was successfully created.

6.1.6.2 Error return codes

Code	Message	Description
400	Bad Request	The data given as parameter is not sufficient to handle the request.
401	Unauthorized	The given API key is not valid.
403	Forbidden	The role of the user authenticated by the API key is not permitted to access the requested endpoint.
404	Not Found	The requested data was not found in the data repository.
601	Invalid ID	Specified item identifier within a POST request is not unique. Depending on the implementation, this can be avoided if the API corrects the ID on demand and returns the data with the correct ID.

6.1.7 Data model and data format

The data used in the SpecIF Web API complies with the SpecIF JSON-schema described above. Currently only JSON is defined as SpecIF data set. SpecIF Web API requests shall accept all data as `application/json` and respond with `application/json` data only.

6.1.8 Parameters

Parameter data in SpecIF Web API requests may be transmitted in three different ways:

- As part of the endpoint URL, defined in curly brackets (e.g. `/specif/v1.1/resources/{id}`), called path-parameters.
- As query parameter appended to the URL such as `?parameterA=1234;parameterB=foe`. Query parameters are optional!
- As body content in JSON format.

The endpoint parameters are defined below, and it is described for each endpoint which parameters are available and how they shall be provided.

6.2 Handling of revisions

Some endpoints have a path parameter `id` to query for data. The revision parameter is defined as an optional query parameter to specify a specific revision. If the revision parameter is missing, the element with the newest change date (`changedAt`) is used.

6.2.1 Revisioning for data elements

A POST request shall always create a new element. If an element with a specified ID already exists, the API has to change the ID and return it in the result body data as first revision (no replaces entries).

To create a new revision of an existing element a PUT request shall be used. The data sent as body parameter in a PUT request shall have the same `id` as the current element.

We have to differentiate the following cases: * If the `id`, `revision` and `replaces` value is identical to an existing element, the API has to create a new revision identifier and add the data as new revision to the existing element. * If the `id` and `replaces` values are set to valid values, the API shall check if the `revision` value is unique and if not, create a new revision and add the data as new revision to the elements that are referenced by the `replaces` values.

6.2.2 Revisioning for meta elements

The versioning of the `data`- and `class`- elements (metatypes) should be handled a little bit differently to the handling of data elements. Such meta elements are typically defined by an administrator in a process of term and vocabulary definitions using SpecIF. It is not required to track each single change on these elements. So the behavior for revisioning is a little bit differently to the element representing concrete PLM-data:

- When a POST request is used, a new element should be created with an initial revision.
- When a PUT request is used with `id` and `revision` set to an existing element and the `replaces` submitted is equal to the existing element, the element data should be updated WITHOUT creating a new revision.
- When a PUT request is used and the `id` and `replaces` is pointing to an existing element, a new revision shall be created. The API has to enforce the consistency and correctness of the `revision` property.

6.3 Data definition endpoints

The following sections describe the SpecIF Web API endpoints for data definition (metadata) endpoints.

6.3.1 Data types

Endpoints for CRUD operations on SpecIF *DataType* elements.

Endpoint	Parameters	Response data	Minimum access role	Description
GET /specif/v1.1/dataTypes	-	DataType[]	Anybody or Reader	Returns all data types with all available revisions.
POST /specif/v1.1/dataTypes	DataType (body)	DataType	Administrator	Create a new DataType and returns the created DataType as response body.
PUT /specif/v1.1/dataTypes	DataType (body)	DataType	Administrator	Update the data type; the supplied ID must exist. The updated data is returned as response body.
GET /specif/v1.1/dataTypes/{id}	<i>id:string</i> (path): The id of the element to get. <i>get.revision:string</i> (query): The revision of the element to get.	DataType	Anybody or Reader	Returns the data type with the given ID.
DELETE /specif/v1.1/dataTypes/{id}	<i>id:string</i> (path): The id of the element to delete. <i>delete.revision:string</i> (query): The revision of the element to delete.	-	Administrator	Delete the data type; the supplied ID must exist. Return an error if there are dependant model elements.
GET /specif/v1.1/dataTypes/{id}/revisions	<i>id:string</i> (path): The id of the element to get.	DataType[]	Anybody or Reader	Returns all element revisions for the given id. These elements have the same id value, but different revisions.

6.3.2 Property classes

Endpoints for CRUD operations on SpecIF *PropertyClass* elements.

Endpoint	Parameters	Response data	Minimum access role	Description
GET /specif/v1.1/propertyClasses	-	PropertyClass[]	Anybody or Reader	Returns all property



Endpoint	Parameters	Response data	Minimum access role	Description
				classes with all available revisions.
POST /specif/v1.1/propertyClasses	PropertyClass (body)	PropertyClass	Administrator	Creates a new DataType and returns the created PropertyClass as response body.
PUT /specif/v1.1/propertyClasses	PropertyClass (body)	PropertyClass	Administrator	Updates the data type; the supplied ID must exist. The updated data is returned as response body.
GET /specif/v1.1/propertyClasses/{id}	<i>id:string</i> (path): The ID of the element to get. <i>revision:string</i> (query): The revision of the element to get.	PropertyClass	Anybody or Reader	Returns the property class with the given ID.
DELETE /specif/v1.1/propertyClasses/{id}	<i>id:string</i> (path): The ID of the element to delete. <i>revision:string</i> (query): The revision of the element to delete.	-	Administrator	Delete the property class; the supplied ID must exist. Return an error if there are dependant model elements.
GET /specif/v1.1/propertyClasses/{id}/revisions	<i>id:string</i> (path): The ID of the element to get.	PropertyClass[]	Anybody or Reader	Returns all element revisions for the given ID. These elements have the same ID value, but different revisions.

6.3.3 Resource classes

Endpoints for CRUD operations on SpecIF *ResourceClass* elements.

Endpoint	Parameters	Response data	Minimum access role	Description
GET /specif/v1.1/resourceClasses	-	ResourceClass[]	Anybody or Reader	Returns all resource classes with all available revisions.
POST /specif/v1.1/resourceClasses	ResourceClass (body)	ResourceClass	Administrator	Creates a new DataType and returns the created ResourceClass as response body.

Endpoint	Parameters	Response data	Minimum access role	Description
PUT /specif/v1.1/resourceClasses	ResourceClass (body)	ResourceClass	Administrator	Updates the data type; the supplied ID must exist. The updated data is returned as response body.
GET /specif/v1.1/resourceClasses/{id}	<i>id:string</i> (path): The ID of the element to get. <i>revision:string</i> (query): The revision of the element to get.	ResourceClass	Anybody or Reader	Returns the resource class with the given ID.
DELETE /specif/v1.1/resourceClasses/{id}	<i>id:string</i> (path): The ID of the element to delete. <i>revision:string</i> (query): The revision of the element to delete.	-	Administrator	Deletes the resource class; the supplied ID must exist. Return an error if there are dependant model elements.
GET /specif/v1.1/resourceClasses/{id}/revisions	<i>id:string</i> (path): The ID of the element to get.	ResourceClass[]	Anybody or Reader	Returns all element revisions for the given ID. These elements have the same ID value, but different revisions.

6.3.4 Statement classes

Endpoints for CRUD operations on SpecIF *StatementClass* elements.

Endpoint	Parameters	Response data	Minimum access role	Description
GET /specif/v1.1/statementClasses	-	StatementClass[]	Anybody or Reader	Returns all statement classes with all available revisions.
POST /specif/v1.1/statementClasses	StatementClass (body)	StatementClass	Administrator	Creates a new StatementClass and returns the created StatementClass as response body.
PUT /specif/v1.1/statementClasses	StatementClass (body)	StatementClass	Administrator	Updates the data type; the supplied ID must exist. The updated data is returned as response body.
GET /specif/v1.1/statementClasses/{id}	<i>id:string</i> (path): The ID of the element to get. <i>revision:string</i> (query): The revision of the element to get.	StatementClass	Anybody or Reader	Returns the resource class with the given ID.
DELETE /specif/v1.1/statementClasses/{id}	<i>id:string</i> (path): The ID of the element to delete. <i>revision:string</i> (query): The revision of the element to delete.	-	Administrator	Deletes the statement class; the supplied ID must exist. Return an error if there are dependant model elements.
GET /specif/v1.1/statementClasses/{id}/revisions	<i>id:string</i> (path): The ID of the element to get.	StatementClass[]	Anybody or Reader	Returns all element revisions for the given ID. These elements have the same ID value, but different revisions.

6.4 Data endpoints

The following sections describe the SpecIF Web API endpoints for data endpoints.

6.4.1 Resources

Endpoints for CRUD operations on SpecIF *Resource* elements.

Endpoint	Parameters	Response data	Minimum access role	Description
GET /specif/v1.1/resources	<i>projectID:string</i> (query): An optional project ID. The endpoint then returns only resources for the given project.	Resource[]	Reader	Returns all resource elements with all available revisions.
POST /specif/v1.1/resources	<i>Resource</i> (body): The resource data to add. <i>projectID:string</i> (query): The optional project ID. If a project ID is not given, the data is added to a default project.	Resource	Editor	Creates a new resource and returns the created resource as response body.
PUT /specif/v1.1/resources	<i>Resource</i> (body): The resource to update.	Resource	Editor	Updates the data type; the supplied ID must exist. The updated data with a new revision is returned as response body.
GET /specif/v1.1/resources/{id}	<i>id:string</i> (path): The ID of the element to get. <i>revision:string</i> (query): The revision of the element to get.	Resource	Reader	Returns the resource with the given ID.
DELETE /specif/v1.1/resources/{id}	<i>id:string</i> (path): The ID of the element to delete. <i>revision:string</i> (query): The revision of the element to delete.	-	Manager	Deletes the resource; the supplied ID must exist.
GET /specif/v1.1/resources/{id}/revisions	<i>id:string</i> (path): The ID of the element to get.	Resource[]	Reader	Returns all element revisions for the given ID. These elements have the same ID value, but different revisions.

6.4.2 Statements

Endpoints for CRUD operations on SpecIF *Statement* elements.

Endpoint	Parameters	Response data	Minimum access role	Description
GET /specif/v1.1/statements	<i>projectID:string</i> (query): The optional project ID to filter statements by <i>project.subjectID:string</i> (query): The optional subject ID to filter statements only sourcing the element with the given ID. <i>subjectRevision:string</i> (query): An optional subject revision. Only useful together with subject ID. <i>objectID:string</i> (query): The optional object ID to filter statements only targeting the element with	Statement[]	Reader	Returns all resource elements with all available revisions.

Endpoint	Parameters	Response data	Minimum access role	Description
	the given ID. <i>objectRevision:string</i> (query): An optional object revision. Only useful together with object ID.			
POST /specif/v1.1/statements	<i>Statement</i> (body): The statement to add.	Statement	Editor	Creates a new statement and returns the created statement as response body.
PUT /specif/v1.1/statements	<i>Statement</i> (body): The statement to update.	Statement	Editor	Updates the data type; the supplied ID must exist. The updated data with a new revision is returned as response body.
GET /specif/v1.1/statements/{id}	<i>id:string</i> (path): The ID of the element to get. <i>revision:string</i> (query): The revision of the element to get.	Statement	Reader	Returns the resource with the given ID.
DELETE /specif/v1.1/statements/{id}	<i>id:string</i> (path): The ID of the element to delete. <i>revision:string</i> (query): The revision of the element to delete.	-	Manager	Delete the statement; the supplied ID must exist.
GET /specif/v1.1/statements/{id}/revisions	<i>id:string</i> (path): The ID of the element to get.	Statement[]	Reader	Returns all element revisions for the given ID. These elements have the same ID value, but different revisions.

6.4.3 Hierarchies

Endpoints for CRUD operations on SpecIF *Hierarchy/Node* elements.

Endpoint	Parameters	Response data	Minimum access role	Description
GET /specif/v1.1/hierarchies	<i>projectID:string</i> (query): An optional project ID. The endpoint then returns only hierarchies for the given <i>project.rootNodesOnly:bool=true</i> (query): If set to true only the hierarchy root nodes are returned and not the complete tree structure.	Node[]	Reader	Returns all hierarchies.
POST /specif/v1.1/hierarchies	<i>Node</i> (body): The node data to add. <i>parent:string</i> (query): An	Node	Editor	Creates a hierarchy (sub-tree) with supplied

Endpoint	Parameters	Response data	Minimum access role	Description
	optional parent node ID. The sub-tree will be inserted as first child. <i>predecessor:string</i> (query): An optional predecessor node ID. The sub-tree will be inserted after the specified node. <i>projectId:string</i> (query): The project ID. If the ID is given, the new hierarchy will be added to the specific project. Only useful for new hierarchies - no parent or predecessor given.			nodes; the supplied ID must be unique. If no ID is supplied, it is generated before insertion. Query <i>?parent=nodeId</i> - the sub-tree will be inserted as first child; query <i>?predecessor=nodeId</i> - the sub-tree will be inserted after the specified node; no query - the sub-tree will be inserted as first element at root level. Without query string, the node (sub-tree) is inserted as first element at root level.
PUT /specif/v1.1/hierarchies	<i>Node</i> (body): The node to update. <i>parent:string</i> (query): An optional parent node ID. The sub-tree will be inserted as first child. <i>predecessor:string</i> (query): An optional predecessor node ID. The sub-tree will be inserted after the specified node.	Node	Editor	Updates the hierarchy; the supplied ID must exist. The updated data is returned as response body.
GET /specif/v1.1/hierarchies/{id}	<i>id:string</i> (path): The ID of the element to get. <i>depth:integer</i> (query): The maximum depth of child nodes to return. If not set the complete hierarchy depth is returned.	Node	-	Returns the node with the given ID.
DELETE /specif/v1.1/hierarchies/{id}	<i>id:string</i> (path): The ID of the element to delete.	-	Editor	Deletes the node with the given ID; the supplied ID must exist. All sub-nodes are deleted as well.

6.4.4 Projects

Endpoints for CRUD operations on SpecIF *Project* elements. Projects in SpecIF are equal to a SpecIF file as defined by the SpecIF-JSON-schema. The SpecIF Web API endpoints for project operations can be used to import or export SpecIF data from and to JSON-files.

Endpoint	Parameters	Response data	Minimum access role	Description
GET /specif/v1.1/projects	-	SpecIF without data and data type arrays.	Reader	Returns all projects; to limit the size only root properties are delivered.
POST /specif/v1.1/projects	<i>SpecIF</i> (body): The SpecIF data to add. <i>integrationID:string</i> (query): If this value is set and a project with	SpecIF	Manager	Creates a new SpecIF project and returns the result as response body.



Endpoint	Parameters	Response data	Minimum access role	Description
	such an ID exists, the data is not added to a new project, but added as new data to the existing project.			
PUT /specif/v1.1/projects	<i>SpecIF</i> (body): The SpecIF data to update.	Manager	SpecIF	Updates a project with the given ID by including the data into an existing project. The project with the supplied ID must exist.
GET /specif/v1.1/projects/{id}	<i>id:string</i> (path): The project ID. <i>hierarchyFilter:list of string</i> (query): An optional comma separated list of hierarchy root node IDs to limit the output of selected hierarchies. <i>includeMetadata:bool=true</i> (query): Set to true if the metadata should be included (e.g. resource classes etc.)	SpecIF	Reader	Gets the project data with the given ID.
DELETE specif/v1.1/projects/{id}	<i>id:string</i> (path): The ID of the element to delete.	-	Administrator	Deletes a project with the given ID.

6.4.5 Files

Endpoints for CRUD operations on SpecIF *File* elements. File elements have a file and an additional description JSON-part. The JSON file description contains a URL where the described file can be accessed using a GET <fileURL> request. If the URL is a relative URL, the file can be accessed adding the relative URL to the prefix GET /specif/v1.1/ The SpecIF Web API is responsible for file management and file storage.

Endpoint	Parameters	Response data	Minimum access role	Description
GET /specif/v1.1/files	<i>projectID:string</i> (query): An optional project ID. The endpoint then returns only file-data for the given project.	JSON File object	Reader	Returns all file descriptions for all available files in all revisions.
POST /specif/v1.1/files	<i>file:binary</i> (body): The file to include to the SpecIF repository. <i>projectID:string</i> (query): An optional project ID.	JSON-File description.	Editor	Creates a new file element.
PUT /specif/v1.1/files/{id}	<i>id:string</i> (path): The file ID as given by the JSON-File object. <i>file:binary</i> (body): The file data.	JSON-File description.	Editor	Updates an existing file.
GET /specif/v1.1/files/{id}	<i>id:string</i> (path): The file ID as given by the JSON-File object. <i>revision:string</i> (query): Optional file revision.	The file content.	Reader	Gets the file content by identifier.
DELETE /specif/v1.1/files/{id}	<i>id:string</i> (path): The file ID to be delete.	-	Administrator	Deletes the file with the given ID.
GET /specif/v1.1/files/{id}/revisions	<i>id:string</i> (path): The file ID to get.	File[]	Reader	Returns all File description revisions for the given ID. These elements have the same ID value, but different revisions.

7 SpecIF Class Definitions

To define the semantics for SpecIF, a collection of data types and class definitions is defined and part of the SpecIF standard.

The class definitions, provided as SpecIF files, are available here:
<https://github.com/GfSE/SpecIF-Class-Definitions>.

It is strongly recommended to use exactly the data type and class definitions described here to get a SpecIF file or SpecIF data set that follows the official SpecIF standard. Of course you can define your own data types and classes to extend the standardized SpecIF semantics with your own domain-specific definitions. Nevertheless, try to reuse the standardized types as often as possible or use the concept of inheritance to define your own types based on existing and standardized ones. This ensures compatibility with the SpecIF standard and allows a semantically correct interpretation of your data by tools supporting SpecIF.

7.1 Domains

The definition of SpecIF classes is organized in application domains. This allows the definition of releases for some domains at the same time, while other domains are still under discussion and development and will be released at a later date.

For SpecIF 1.1 three domains are selected to be part of the first release:

- 01 - Base Definitions,
- 02 - Requirements Engineering,
- 03 - Model Integration.

The other domains are still under discussion and not yet part of an official SpecIF release. It is planned to release them with a future release.

7.1.1 Domain types

The following list shows the currently defined domains and their IDs:

Domain ID	Domain	Description	Release status
01	Base definitions	Common definitions relevant for all domains (e.g. primitive data types).	Released in 1.1
02	Requirements Engineering	Classical requirements engineering following the IREB definitions.	Released in 1.1
03	Model Integration	SpecIF mapping for the Fundamental Modeling Concepts approach usable to integrate system modeling data.	Released in 1.1
04	Automotive Requirements Engineering	Automotive-specific requirements engineering extensions (VDA).	Unreleased
05	Agile Requirements Engineering	Requirements engineering for agile development (e.g. epics and user stories).	Unreleased
06	UML/SysML Integration	Deprecated. Covered by domain 03-Model Integration.	Deprecated
07	Issue Management	Issue and Task management.	Unreleased
08	BOM	Bill of materials.	Unreleased
09	Variant Management	Feature model-based variant management.	Unreleased
10	Vocabulary Definition	Resources to define Vocabularies (e.g. SpecIF Vocabulary).	Unreleased
11	Testing	Testing domain according to ISTQB definitions.	Unreleased

7.2 Domain 01: Base Definitions

7.2.1 Data types of domain 01: Base Definitions

title	id	revision	type	description
Boolean	DT-Boolean	1.1	xs:boolean	The Boolean data type.
Byte	DT-Byte	1.1	xs:integer	A byte is an integer value between 0 and 255.
Integer	DT-Integer	1.1	xs:integer	A numerical integer value from -32768 to 32767.
Real	DT-Real	1.1	xs:double	A floating point number (double).
Real with 2 Decimals	DT-Decimal2	1.1	xs:double	A floating point number (double) with two fraction digits.
Date or Timestamp	DT-DateTime	1.1	xs:dateTime	Date or timestamp in ISO-format
String[256]	DT-ShortString	1.1	xs:string	String with max. length 256
Plain or formatted Text	DT-Text	1.1	xs:string	An account of the resource (source: http://dublincore.org/documents/dcmi-terms/). Descriptive text represented in plain or rich text using XHTML. SHOULD include only content that is valid and suitable inside an XHTML <div> element (source: http://open-services.net/).
URL	DT-URL	1.1	xs:string	A uniform resource locator.
E-mail	DT-EmailAddress	1.1	xs:string	Data type to represent an e-mail address.
SpecIF:LifeCycleStatus	DT-LifeCycleStatus	1.1	xs:string	Enumerated values for status SpecIF:LifecycleStatusDeprecated [V-Status-0] SpecIF:LifecycleStatusRejected [V-Status-1] SpecIF:LifecycleStatusInitial [V-Status-2] SpecIF:LifecycleStatusDrafted [V-Status-3] SpecIF:LifecycleStatusSubmitted [V-Status-4] SpecIF:LifecycleStatusApproved [V-Status-5] SpecIF:LifecycleStatusReady [V-Status-8] SpecIF:LifecycleStatusDone [V-Status-6] SpecIF:LifecycleStatusValidated [V-Status-9] SpecIF:LifecycleStatusReleased [V-Status-7] SpecIF:LifecycleStatusWithdrawn [V-Status-10]
SpecIF:Priority	DT-Priority	1.1	xs:string	Enumerated values for priority SpecIF:priorityHigh [V-Prio-0] SpecIF:priorityRatherHigh [V-Prio-1] SpecIF:priorityMedium [V-Prio-2] SpecIF:priorityRatherLow [V-Prio-3] SpecIF:priorityLow [V-Prio-4]
SpecIF:Discipline	DT-Discipline	1.1	xs:string	Enumerated values for engineering discipline SpecIF:DisciplineSystem [V-discipline-4] SpecIF:DisciplineMechanics [V-discipline-0] SpecIF:DisciplineElectronics [V-discipline-1] SpecIF:DisciplineSoftware [V-discipline-2] SpecIF:DisciplineSafety [V-discipline-3]

7.2.2 Property classes of domain 01: Base Definitions

title	id	revision	data Type	description
dcterms:identifier	PC-VisibleId	1.1	String[256]	A unique reference to the resource within a given context. (source: DCMI)



title	id	revision	dataType	description
				An identifier for a resource. This identifier may be unique with a scope that is defined by the RM provider. Assigned by the service provider when a resource is created. Not intended for end-user display. (source: OSLC)
dcterms:title	PC-Name	1.1	String[256]	A name given to the resource. (source: DCMi) Title (reference: Dublin Core) of the resource represented as rich text in XHTML content. SHOULD include only content that is valid inside an XHTML element. (source: OSLC)
dcterms:description	PC-Description	1.1	Plain or formatted Text	An account of the resource. (source: DCMi) Descriptive text (reference: Dublin Core) about resource represented as rich text in XHTML content. SHOULD include only content that is valid and suitable inside an XHTML <div> element. (source: OSLC)
SpecIF:Origin	PC-Origin	1.1	String[256]	The origin (source, reference) of an information or requirement.
SpecIF:Diagram	PC-Diagram	1.1	Plain or formatted Text	A partial graphical representation (diagram) of a model.
SpecIF:Notation	PC-Notation	1.1	String[256]	The notation used by a model diagram, e.g. 'BPMN:2.0', 'OMG:SysML:1.3:Activity Diagram' or 'FMC:Block Diagram'.
SpecIF:LifeCycleStatus	PC-LifeCycleStatus	1.1	SpecIF:LifeCycleStatus	The 'status', e.g. lifecycle state, of the resource.
SpecIF:Priority	PC-Priority	1.1	SpecIF:Priority	The 'priority' of the resource.
SpecIF:Discipline	PC-Discipline	1.1	SpecIF:Discipline	The engineering discipline (system, electronics, mechanics, software, safety).
SpecIF:Responsible	PC-Responsible	1.1	String[256]	The 'person' being responsible for the resource.
SpecIF:DueDate	PC-DueDate	1.1	Date or Timestamp	A 'due date' for the resource.
UML:Stereotype	PC-Stereotype	1.1	String[256]	A stereotype gives an element an additional/different meaning.
SpecIF:Abbreviation	PC-Abbreviation	1.1	String[256]	An abbreviation for the resource or statement.
dcterms:type	PC-Type	1.1	String[256]	The element type resp. the metamodel element (e.g. OMG:UML:2.5.1:Class)
SpecIF:Alias	PC-Alias	1.1	String[256]	An alias name for the resource.
rdf:value	PC-Value	1.1	Plain or formatted Text	A value of different meaning, depending on the element type (attribute default value, a taggedValue value etc.).

7.2.3 Resource classes of domain 01: Base Definitions

title	id	revision	description
SpecIF:Heading	RC-Folder	1.1	Folders with title and text for chapters or descriptive paragraphs. Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] dcterms:type [PC-Type 1.1]
SpecIF:Paragraph	RC-Paragraph	1.1	Information with text for descriptive paragraphs. Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1]
SpecIF:Hierarchy	RC-Hierarchy	1.1	Root node of a hierarchically organized specification (outline). Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1]
SpecIF:Comment	RC-Comment	1.1	Comment referring to a model element ('resource' or 'statement' in general). Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1]

7.2.4 Statement classes of domain 01: Base Definitions

title	id	revision	description
rdf:type	SC-Classifier	1.1	States that the relation subject is an instance of the relation object.
SpecIF:refersTo	SC-refersTo	1.1	A resource 'refers to' any other resource. Property classes: dcterms:type [PC-Type 1.1]

7.3 Domain 02: Requirements Engineering

7.3.1 Data types of domain 02: Requirements Engineering

title	id	revision	type	description
IREB:RequirementType	DT-RequirementType	1.1	xs:string	Enumerated values for the requirement type according to IREB. IREB:FunctionalRequirement [V-RequirementType-0] IREB:QualityRequirement [V-RequirementType-1] IREB:Constraint [V-RequirementType-2]
Perspective	DT-Perspective	1.1	xs:string	Enumerated values for the perspective (of a requirement) IREB:PerspectiveBusiness [V-perspective-0] IREB:PerspectiveStakeholder [V-perspective-3] IREB:PerspectiveUser [V-perspective-1] IREB:PerspectiveOperator [V-perspective-4] IREB:PerspectiveSystem [V-perspective-2]

7.3.2 Property classes of domain 02: Requirements Engineering

title	id	revision	data Type	description
IREB:RequirementType	PC-RequirementType	1.1	IREB:RequirementType	Enumerated value for the requirement type according to IREB.

title	id	revision	dataType	description
SpecIF:Perspective	PC-Perspective	1.1	Perspective	Enumerated values for the perspective (of a requirement).

7.3.3 Resource classes of domain 02: Requirements Engineering

title	id	revision	description
IREB:Requirement	RC-Requirement	1.1	<p>A 'requirement' is a singular documented physical and functional need that a particular design, product or process must be able to perform.</p> <p>Property classes:</p> <p>dcterms:identifier [PC-VisibleId 1.1] dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] IREB:RequirementType [PC-RequirementType 1.1] SpecIF:Priority [PC-Priority 1.1] SpecIF:LifeCycleStatus [PC-LifeCycleStatus 1.1] SpecIF:Perspective [PC-Perspective 1.1] SpecIF:Discipline [PC-Discipline 1.1]</p>
SpecIF:Feature	RC-Feature	1.1	<p>A 'feature' is an intentional distinguishing characteristic of a system, often a unique selling proposition.</p> <p>Property classes:</p> <p>dcterms:identifier [PC-VisibleId 1.1] dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] SpecIF:Priority [PC-Priority 1.1] SpecIF:LifeCycleStatus [PC-LifeCycleStatus 1.1] SpecIF:Perspective [PC-Perspective 1.1] SpecIF:Discipline [PC-Discipline 1.1]</p>

7.3.4 Statement classes of domain 02: Requirements Engineering

title	id	revision	description
SpecIF:dependsOn	SC-dependsOn	1.1	Statement: Requirement/feature depends on requirement/feature.
SpecIF:duplicates	SC-duplicates	1.1	The subject requirement duplicates the object requirement.
SpecIF:contradicts	SC-contradicts	1.1	The subject requirement contradicts the object requirement.
IREB:refines	SC-refines	1.1	The subject requirement refines the object requirement.

7.4 Domain 03: Model Integration

7.4.1 Data types of domain 03: Model Integration

title	id	revision	type	description
SpecIF:VisibilityKind	DT-VisibilityKind	1.1	xs:string	<p>Enumerated values for visibility.</p> <p>UML:Public [V-VisibilityKind-0] UML:Private [V-VisibilityKind-1] UML:Protected [V-VisibilityKind-2] UML:Package [V-VisibilityKind-3] UML:Internal [V-VisibilityKind-4] UML:ProtectedInternal [V-VisibilityKind-5]</p>

7.4.2 Property classes of domain 03: Model Integration

title	id	revision	dataType	description
SpecIF:Visibility	PC-Visibility	1.1	SpecIF:VisibilityKind	The visibility of a resource (e.g. public,

title	id	revision	dataType	description
				private, protected,...) as known from object orientation.
SpecIF:ImplementationLanguage	PC-ImplementationLanguage	1.1	String[256]	The name of an used implementation language (e.g. C, C++, C#, Java, ADA, OCL, ALF etc.).

7.4.3 Resource classes of domain 03: Model Integration

title	id	revision	description
UML:Package	RC-Package	1.1	A 'package' is used to bring structure into a model. Packages can contain further packages and/or (model-)elements. It corresponds to a package in UML/SysML or a folder in a file system. Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] SpecIF:LifeCycleStatus [PC-LifeCycleStatus 1.1] SpecIF:Visibility [PC-Visibility 1.1] dcterms:type [PC-Type 1.1] UML:Stereotype [PC-Stereotype 1.1] rdf:value [PC-Value 1.1] SpecIF:Alias [PC-Alias 1.1]
SpecIF:Diagram	RC-Diagram	1.1	A 'diagram' is a graphical model view with a specific communication purpose, e.g. a business process or system composition. Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] SpecIF:Diagram [PC-Diagram 1.1] dcterms:type [PC-Type 1.1] SpecIF:Notation [PC-Notation 1.1] SpecIF:LifeCycleStatus [PC-LifeCycleStatus 1.1] UML:Stereotype [PC-Stereotype 1.1]
FMC:Actor	RC-Actor	1.1	An 'actor' is a fundamental model element type representing an active entity, be it an activity, a process step, a function, a system component or a role. Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] SpecIF:LifeCycleStatus [PC-LifeCycleStatus 1.1] SpecIF:Visibility [PC-Visibility 1.1] dcterms:type [PC-Type 1.1] UML:Stereotype [PC-Stereotype 1.1] rdf:value [PC-Value 1.1] SpecIF:Alias [PC-Alias 1.1]
FMC:State	RC-State	1.1	A 'state' is a fundamental model element type representing a passive entity, be it a value, a condition, an information storage or even a physical shape. Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1]

title	id	revision	description
			SpecIF:LifeCycleStatus [PC-LifeCycleStatus 1.1] SpecIF:Visibility [PC-Visibility 1.1] dcterms:type [PC-Type 1.1] UML:Stereotype [PC-Stereotype 1.1] rdf:value [PC-Value 1.1] SpecIF:Alias [PC-Alias 1.1]
FMC:Event	RC-Event	1.1	An 'event' is a fundamental model element type representing a time reference, a change in condition/value or more generally a synchronization primitive. Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] SpecIF:LifeCycleStatus [PC-LifeCycleStatus 1.1] SpecIF:Visibility [PC-Visibility 1.1] dcterms:type [PC-Type 1.1] UML:Stereotype [PC-Stereotype 1.1] rdf:value [PC-Value 1.1] SpecIF:Alias [PC-Alias 1.1]
SpecIF:Collection	RC-Collection	1.1	A 'collection' is a logic (often conceptual) group of resources linked with a SpecIF:contains statement. It corresponds to a 'group' in BPMN diagrams or a 'boundary' in UML diagrams. Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] SpecIF:LifeCycleStatus [PC-LifeCycleStatus 1.1] SpecIF:Visibility [PC-Visibility 1.1] dcterms:type [PC-Type 1.1] UML:Stereotype [PC-Stereotype 1.1] rdf:value [PC-Value 1.1] SpecIF:Alias [PC-Alias 1.1]
SpecIF:SourceCode	RC-SourceCode	1.1	Source code assigned to a model element (e.g. an activity or operation). Typically used for model-based code generation to provide some code snippets inside the model. Property classes: dcterms:title [PC-Name 1.1] SpecIF:ImplementationLanguage [PC-ImplementationLanguage 1.1] rdf:value [PC-Value 1.1]

7.4.4 Statement classes of domain 03: Model Integration

title	id	revision	description
SpecIF:shows	SC-shows	1.1	Statement: Plan resp. diagram shows model element.
SpecIF:contains	SC-contains	1.1	Statement: Model element contains model element.
SpecIF:serves	SC-serves	1.1	Statement: An actor serves an actor.
SpecIF:stores	SC-stores	1.1	Statement: Actor (role, function) writes and reads state (information).
SpecIF:writes	SC-writes	1.1	Statement: Actor (role, function) writes state (information).
SpecIF:reads	SC-reads	1.1	Statement: Actor (role, function) reads state (information).
SpecIF:influences	SC-influences	1.1	Statement: A state (information) influences a state (information).
SpecIF:precedes	SC-precedes	1.1	An FMC:Actor 'precedes' an FMC:Actor or an FMC:Actor 'precedes' an FMC:Event or an FMC:Event 'precedes' an



title	id	revision	description
			FMC:Actor. The rdf:type property specifies if it is a simple precedes, a SpecIF:signals or a SpecIF:triggers. Property classes: dcterms:type [PC-Type 1.1]
oslc_rm:satisfies	SC-satisfies	1.1	Statement: Model element satisfies requirement.
SpecIF:allocates	SC-allocates	1.1	Statement: Model element is allocated to model element. The semantics are equal to allocation in SysML or deployment relation in UML.
SpecIF:implements	SC-implements	1.1	A FMC:Actor or FMC:State ‘implements’ a SpecIF:Feature.
SpecIF:isAssociatedWith	SC-isAssociatedWith	1.1	The subject is associated with the object. Property classes: dcterms:type [PC-Type 1.1]
SpecIF:isSpecializationOf	SC-isSpecializationOf	1.0	A term is a specialization of another, such as ‘Passenger Car’ and ‘Vehicle’.

8 Introduction to SpecIF Model Integration

The usage of SpecIF for a specific purpose is called an ‘application’. An important application is to integrate models and other specification artifacts from different sources. Currently transformations or importers exist for BPMN, Archimate, ReqIF, Excel and UML/SysML.

The integration of elements from different sources is generally done by name and type. In other words, two elements in different models/notations are considered the *same*, if their names and types are equal. Checking for equal names is rather simple, but checking for equal types is almost impossible, since every notation and tool is using a different set of types.

Therefore an abstraction (or mapping) to three *fundamental* model-element types is performed, before checking for equality. The model element types of the *Fundamental Modeling Concepts (FMC)* have been selected for this purpose, namely *Actor*, *State* and *Event*. Model elements used by any method or notation can be mapped to these [[Dungern2016](#)].

8.1 Model Integration Resources

There are just a few SpecIF resource classes used for semantic model integration.

8.1.1 Fundamental Model Element Types

As mentioned before, the Fundamental Modeling Concepts (FMC) approach defines three fundamental element types to represent all semantic aspects:

- An ■ *Actor* (vocabulary term FMC:Actor) is a fundamental model element type representing an active entity, be it an activity, a process step, a function, a system component or a user role.
- A ● *State* (vocabulary term FMC:State) is a fundamental model element type representing a passive entity, be it a value, an information store, even a color or shape.
- An ♦ *Event* (vocabulary term FMC:Event) is a fundamental model element type representing a time reference, a change in condition/value or more generally a synchronization primitive.

The idea behind using just these three elements is, that all kinds of structural or behavioral modeling can be expressed as a bipartite graph of just these elements. An example taken from structural modeling are Component Diagrams in UML or Internal Block Diagrams in SysML.

As an example of behavioral modeling, Petri-Nets consist of transitions (actors) and places (states).

Hereby the components are active elements (represented as SpecIF:Actor). Because the components are active system elements, they are able to change the system state. Passive elements (represented as SpecIF:State) may represent data, like UML-object-elements, whose attributes are mapped to state elements, because they represent a current system state by their attribute values.

While all model elements are mapped to the three fundamental elements, the original model element type is kept by the property *dcterms:type*. Storing the original model element in the *dcterms:type* property is also important for bidirectional transformations.


8.1.2 Requirement and Feature

As extension to the fundamental modeling elements and the collection two further elements are defined in SpecIF: The resource classes *Requirement* and *Feature*.

- A ★ *Feature* (vocabulary term SpecIF:Feature) is an intentional distinguishing characteristic of a system, often a so-called ‘Unique Selling Proposition’.
- A ♡ *Requirement* (vocabulary term IREB:Requirement) is a singular documented physical and functional need that a particular design, product or process must be able to perform.


These elements are widely used in Systems Engineering of all kinds of systems and are in fact highly important in practice. So these elements complement the set of fundamental elements used to express contents required for system modeling in the Product Lifecycle Management.

8.1.3 View

In many engineering models the elements defining a concept or documentation are visualized graphically. Such visualizations are called a view or diagram. SpecIF defines the resource class  *Diagram* (vocabulary term SpecIF:Diagram) to represent all kinds of graphical visualizations of model data. All diagrams of all kinds of graphical notations can be visualized applying this resource class.


8.1.4 Package

Models can contain many elements and diagrams. To structure and navigate through the elements used in a model the concept of folders or packages is often used to bring structure to a model. This structure is normally used independently from semantics, which are expressed by the diagrams using model elements and connectors. The model structure is used for navigation and model data organization to sort elements or group them.

In SpecIF the concept of packages is taken from UML and provided by the resource class  *Package* (vocabulary term UML:Package). Applications for package resources are packages in UML/SysML or folders/directories in a file system etc.

Because a package can contain other packages, a hierarchy structure (or ‘tree’) can be expressed using SpecIF.

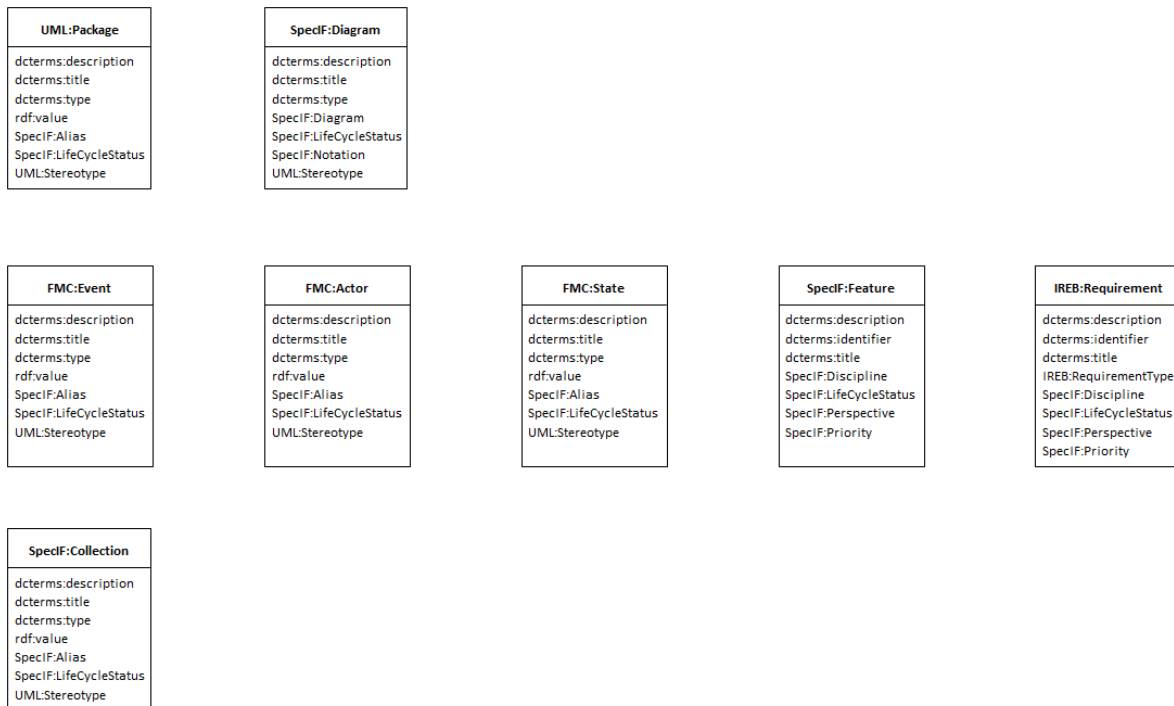
8.1.5 Collection

Models can express a logical or conceptual structure of modeled aspects or the model itself by grouping elements together. This aspect is covered by the resource class  *Collection* (vocabulary term SpecIF:Collection). Examples for collections are groups in BPMN or boundary elements on UML diagrams.

A package is used to bring structure into a model without expressing some semantic aspects. It is just for sorting elements and helps the users of the model to find things fast and easy.

In comparison to a package a collection is typically defined as a graphical element on a diagram to express that the element surrounded by the collection element are grouped together or have a similar meaning or responsibility.

8.1.6 A glimpse on the elements of SpecIF Model Integration



Model Integration resources

The (class-)diagram above gives an overview of the most important element types, defined in SpecIF and used for semantic model integration. This is called the *SpecIF Integration Model*.

Beside the class elements, representing the resource classes, association connections between the classes show the available statements, defined by SpecIF statement classes. For better readability of the class diagrams, the aspects are shown on more than one diagram. You have to interpret the diagrams in combination to get the complete picture of SpecIF Model Integration. If two diagrams show a resource class with the same name, it is the same element and all associations defined by one diagram are also valid for the second diagram.

8.2 Model Integration statements

The statements in SpecIF are used to express a predicate logic (subject - predicate - object) between SpecIF elements. Examples for such logical expressions are:

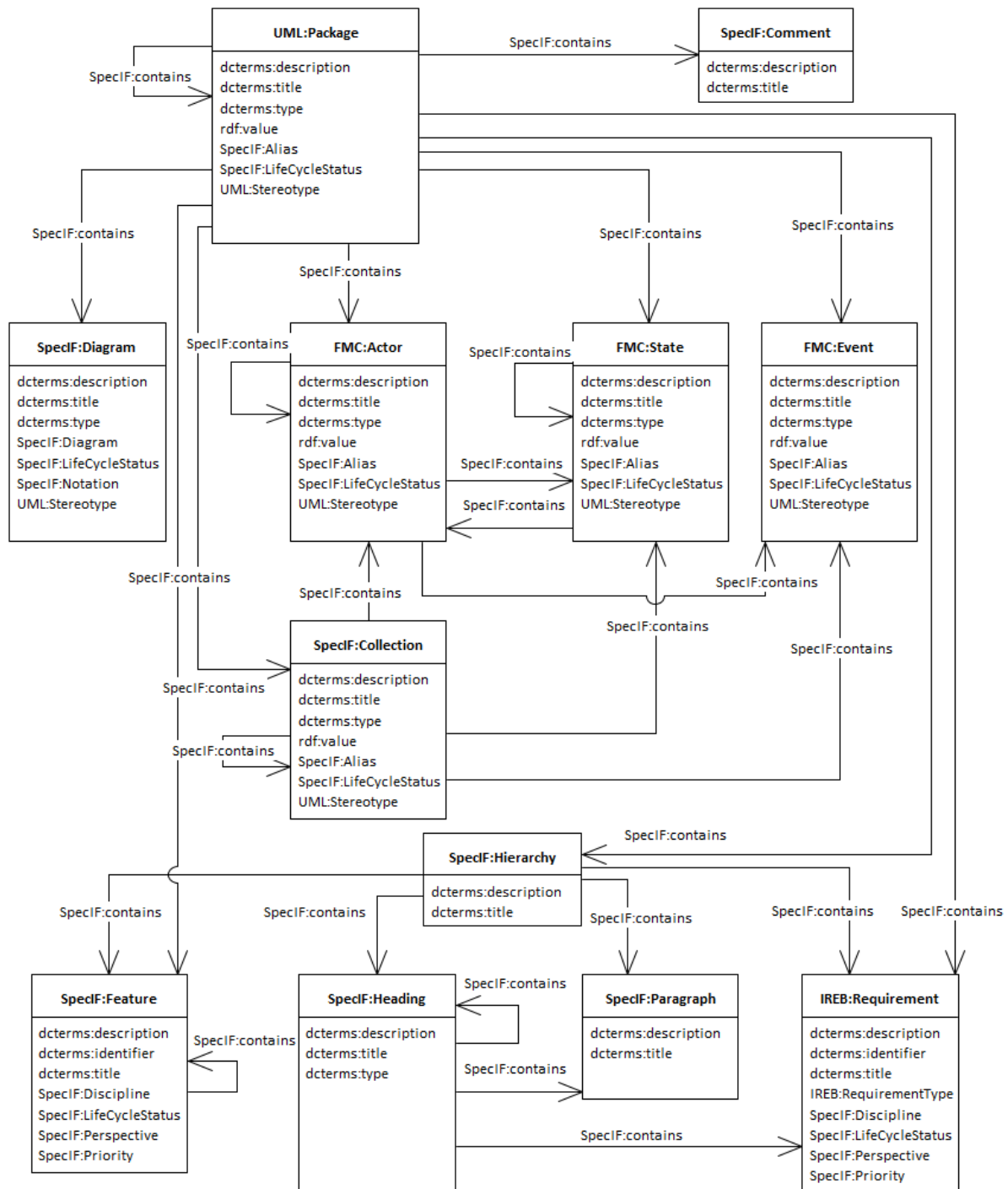
- *A collection contains a diagram.*
- *An event triggers an actor.*
- *A diagram shows a state.*

In the context of model integration we differentiate between statements to express behavioral aspects and structural aspects.

8.2.1 Expressing structure

Structural aspects are expressed with the following statements:

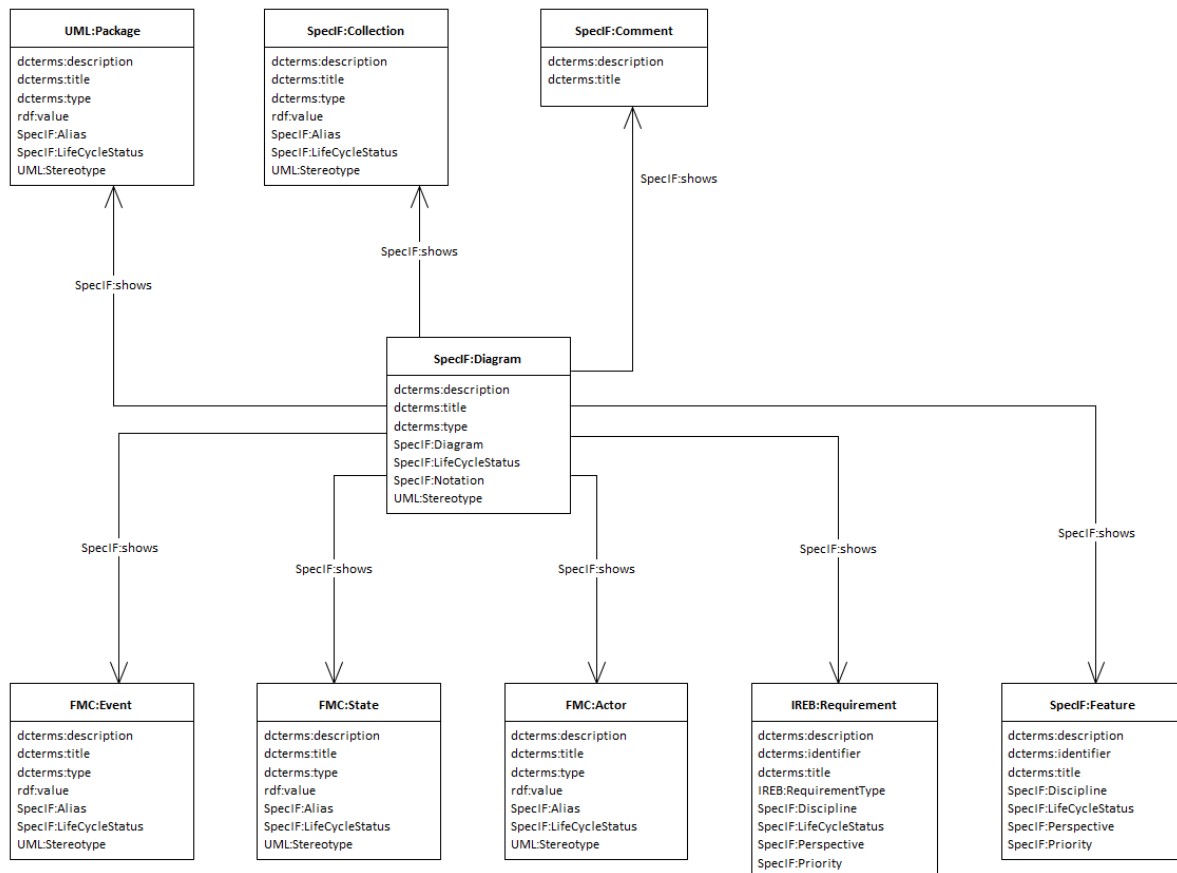
- The *contains* statement expresses that an element is contained in another element. Typical application scenarios are to express for example that a collection *contains* an actor or a state *contains* a sub-state etc.



Application of the contains statement in SpecIF

- The *shows* statement expresses that a model element is depicted on a diagram. This semantic relation is obvious for a human observer of a diagram. If there is the explicit statement it is easily understood by a machine too. Not only are all resources and statements (nodes and edges) of the diagram easily listed, but also the diagrams

showing a given resource or statement. This is known as the concept of separation of model and view.



Application of the shows statement in SpecIF

8.2.2 Expressing traceability aspects

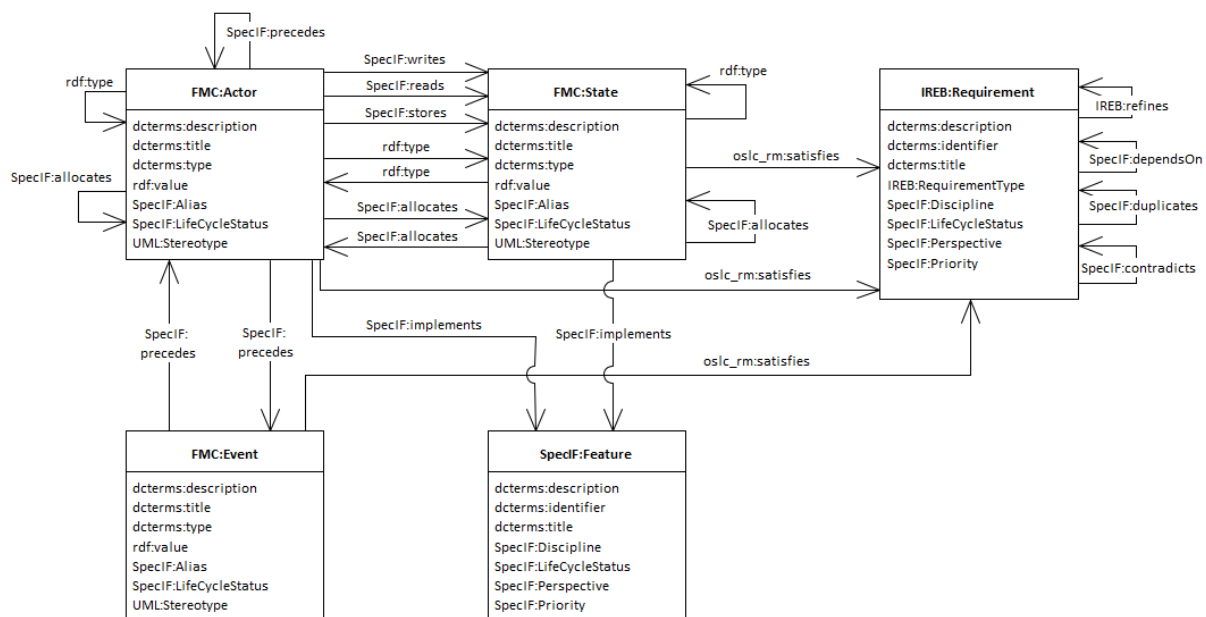
- To express traceability-dependencies between requirement elements and requirements or other element types, the following statements are used:
 - A requirement *refines* a requirement,
 - A requirement *dependsOn* a requirement,
 - A requirement *duplicates* a requirement,
 - A requirement *contradicts* a requirement,
 - A model element (state, actor or event) *satisfies* a requirement.
- The concept of allocation known from systems engineering is mapped to SpecIF by defining the *allocates* statement.

8.2.3 Expressing behavior

To express the behavioral aspects of a system or a process the following statements are available in SpecIF:

- An actor *writes* a state. In a system composition a function writes a value.
- An actor *reads* a state. In a system composition a function reads a value.
- An actor *stores* a state. This is the combination of read and write and equates to a bi-directional data or material exchange.
- An actor *precedes* an actor. This is used to express that a behavior consists of a sequence of actions.
- An actor *precedes* an event with *dcterms:type* set to *SpecIF:signals*. This is used to express that an event is created by an action.
- An event *precedes* an actor with *dcterms:type* set to *SpecIF:triggers*. This is used to express that an action is triggered by an incoming event.

The following diagram shows the behavioral statements used in SpecIF model integration.



Behavioral aspects in SpecIF model integration statements

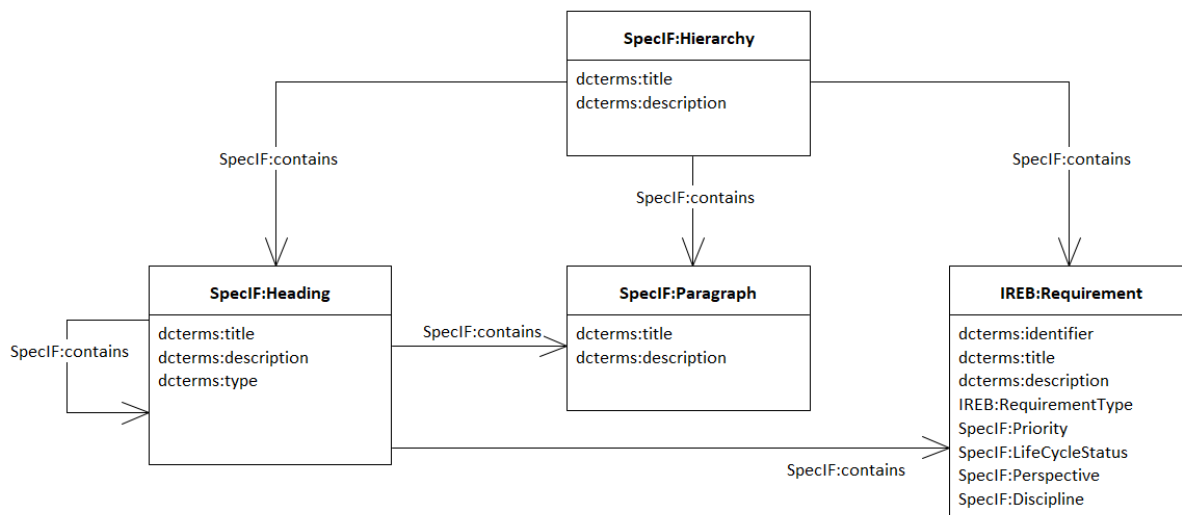
8.2.4 Instantiation

To express that a resource is an instance of another resource (e.g. an object is an instance of a class, a method return type is an instance of a type definition etc.) the statement *rdf:type* is used. This statement expresses that the subject is an instance of the statement object. Use this statement in all cases where class-instance semantics shall be expressed.

8.2.5 Document outlines

SpecIF defines resource class types useful for applications dealing with textual documentation. Typical application scenarios are * Document-based requirement specifications, * User manuals, * Any other kind of textual specifications or text documents.

The following class diagram shows the SpecIF application for document structures.



Resource and statement usage for document structures

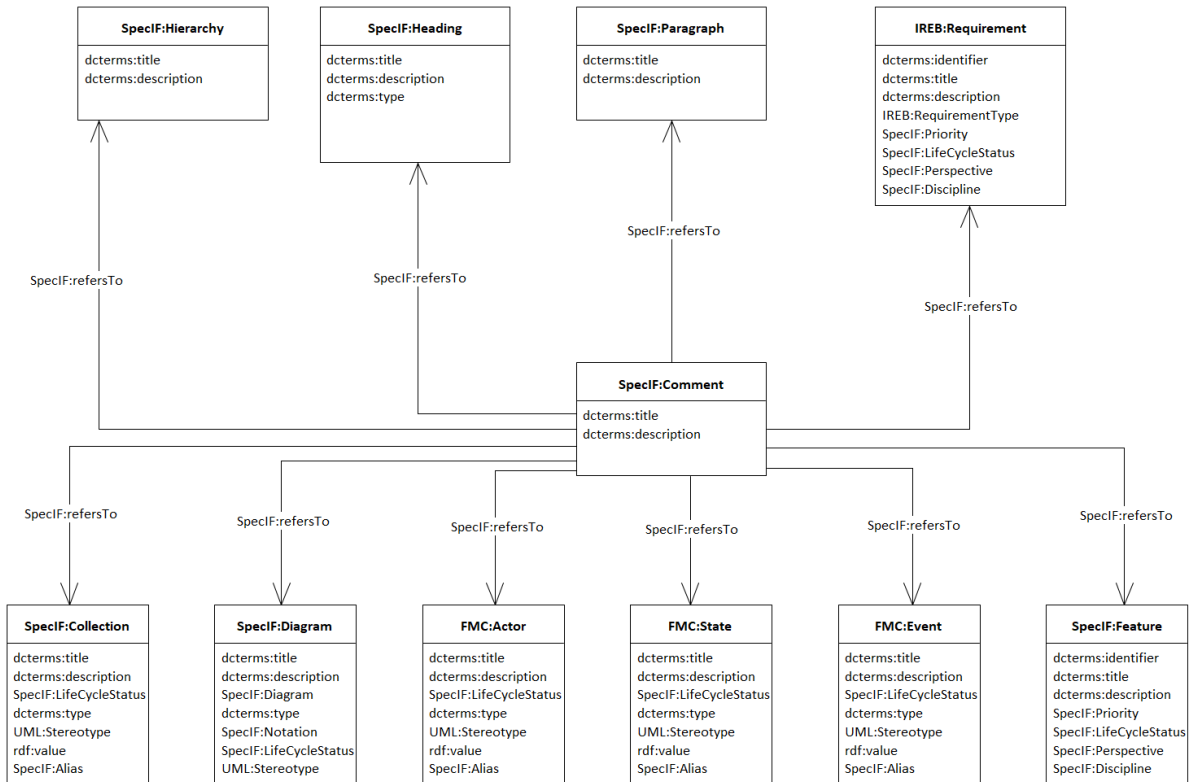
- The **Hierarchy** element is the root for a document,
- A 'H' **Heading** is used to define a heading text and to bring structure into a document,
- A 'P' **Paragraph** can be used to include some text paragraphs with no special semantics (e.g. prose text) into the document,
- A **Feature** and
- a **Requirement** are, as explained above, important to describe benefits and needs of a system to develop.

The statement *contains* is used to define the semantics for the document hierarchy.

8.2.6 Comments

The SpecIF resource type **Comment** is used to include or add additional comments to a resource in a model or a document element. To assign a comment to a resource the statement type *refersTo* is used. Typical application scenarios for using comments are reviews (review comments) or comments attached to a resource as additional short information.

Comments are usually transient: Upon agreement, the model or text is changed accordingly and the comment deleted. When generating a document, the comments can be included or omitted, similarly to MS Word.



Usage of the Comment resource type in SpecIF

8.3 SpecIF Classes for Model Integration

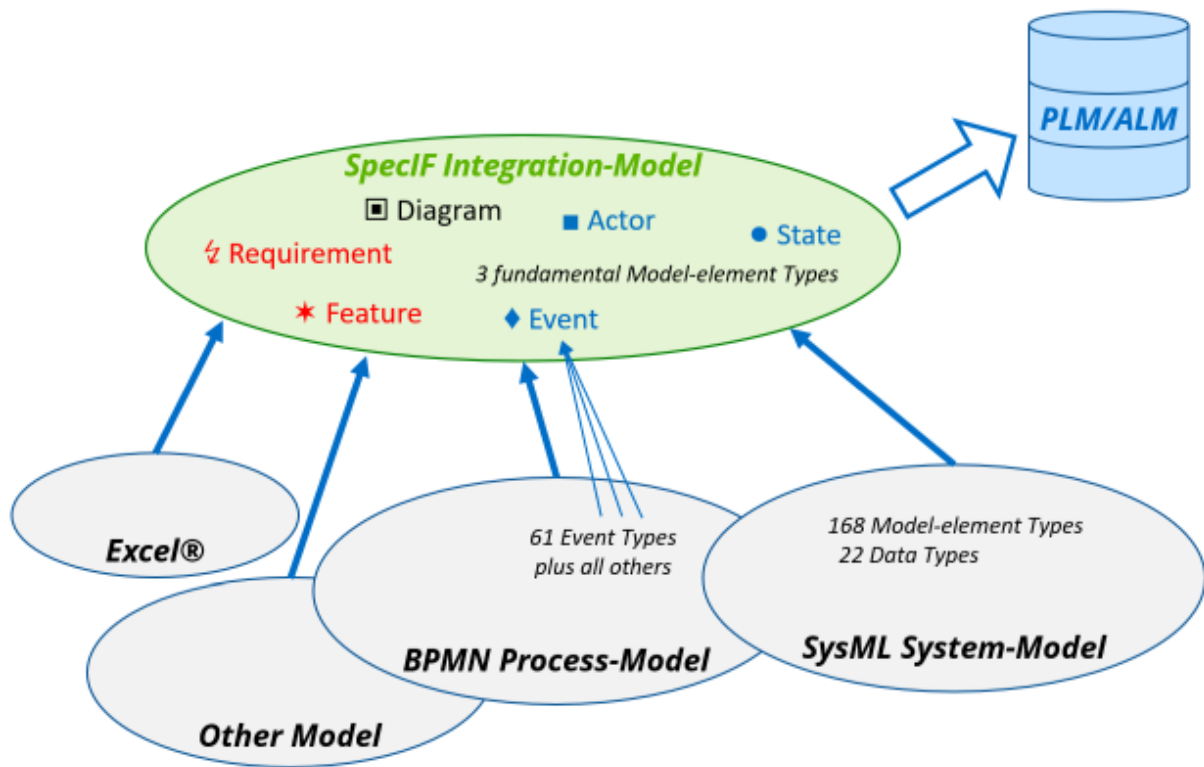
As discussed, the SpecIF schema is generic and allows various applications such as model integration. Each application is characterized by a set of agreed-upon data types and classes. The class definitions described above for integration of models from different tools and notations have been developed over many years through projects and case studies in the arena of Enterprise Architecture (notations FMC, BPMN and Archimate) and in the arena of Mechatronic Systems Engineering (notations FMC and SysML).

With the set of elements defined and released with version 1.1 of SpecIF it is possible to map nearly all elements - existing and used in graphical 2D models and in textual specification and documentation content - over the lifecycle of a system.

With this first release of SpecIF there are surely some gaps for some specialized application scenarios and domains, but the goal of covering a bulk of artifacts existing in PLM and MBSE is still achieved. SpecIF will show its potential as integration and data exchange standard with more upcoming applications and practical use.

8.3.1 Mapping of different modeling environments to SpecIF

Transformations are in fact a mapping from element types of different modeling environments resp. data models to SpecIF. In case of SysML, the mapping relates SysML model element types to SpecIF model element types. The same applies to various other applications such as BPMN and Archimate.



Principle of semantic model integration

The following chapters describe semantic mappings and examples for transformations from different modeling environments to SpecIF. It is currently still work in progress and not yet complete. The first release of SpecIF will therefore not define a fully-complete mapping for each modeling environment, but still tries to define a subset for the most important elements and how to map them to SpecIF.

7. [Model Integration Guide for FMC](#)
8. [Model Integration Guide for BPMN](#)
9. [Model Integration Guide for Archimate](#)
10. [Model Integration Guide for UML and SysML](#)

9 SpecIF Model Integration Guide for ArchiMate®

9.1 ArchiMate® to SpecIF mapping

9.1.1 Resources

ArchiMate® Open Exchange (XML)	SpecIF
BusinessActor, BusinessRole, BusinessCollaboration, BusinessInterface, BusinessProcess, BusinessFunction, BusinessInteraction, BusinessService, ApplicationComponent, ApplicationCollaboration, ApplicationInterface, ApplicationFunction, ApplicationInteraction, ApplicationProcess, ApplicationService, Node, Equipment, Facility, DistributionNetwork, Device, SystemSoftware, TechnologyCollaboration, TechnologyInterface, Path, CommunicationNetwork, TechnologyFunction, TechnologyProcess, TechnologyInteraction, TechnologyService, OrJunction, AndJunction	FMC:Actor
Goal, Capability, Contract, Representation, Artefact, Product, BusinessObject, DataObject	FMC:State
BusinessEvent, ApplicationEvent, TechnologyEvent, ImplementationEvent	FMC:Event
Requirement, Constraint	IREB:Requirement
Location, Grouping	SpecIF:Collection

Where:

- The listed elements are described in the [ArchiMate® 3.1 Specification](#).
- All SpecIF resource class terms are defined in the [Vocabulary](#).
- For all entities in the left column the namespace ‘archimate:’ is used; the XML header defines

```
xsi:schemaLocation="http://www.opengroup.org/xsd/archimate/3.0/
http://www.opengroup.org/xsd/archimate/3.1/archimate3_Diagram.xsd".
```
- The original model element type is stored in a property named *dcterms:type*.

9.1.2 Statements

At present, the following statements are derived from ArchiMate® diagrams, where the *statement terms (predicates)* are highlighted in *italics*:

ArchiMate® Open Exchange (XML)	SpecIF	Comment
view	SpecIF:shows	
Composition, Aggregation, Realization, Assignment	SpecIF:contains	ArchiMate® Structural (or ‘unifying’) Relationship
Access (accessType:Write)	SpecIF:writes	ArchiMate® Dependency Relationship
Access (accessType:Read)	SpecIF:reads	ArchiMate® Dependency Relationship
Association	SpecIF:isAssociatedWith	ArchiMate® Dependency Relationship
Influence	SpecIF:influences	ArchiMate® Dependency Relationship
Serving	SpecIF:serves	ArchiMate® Dependency Relationship
Flow, Triggering	SpecIF:precedes	ArchiMate® Dynamic Relationship
Specialization	SpecIF:isSpecializationOf	

Where:

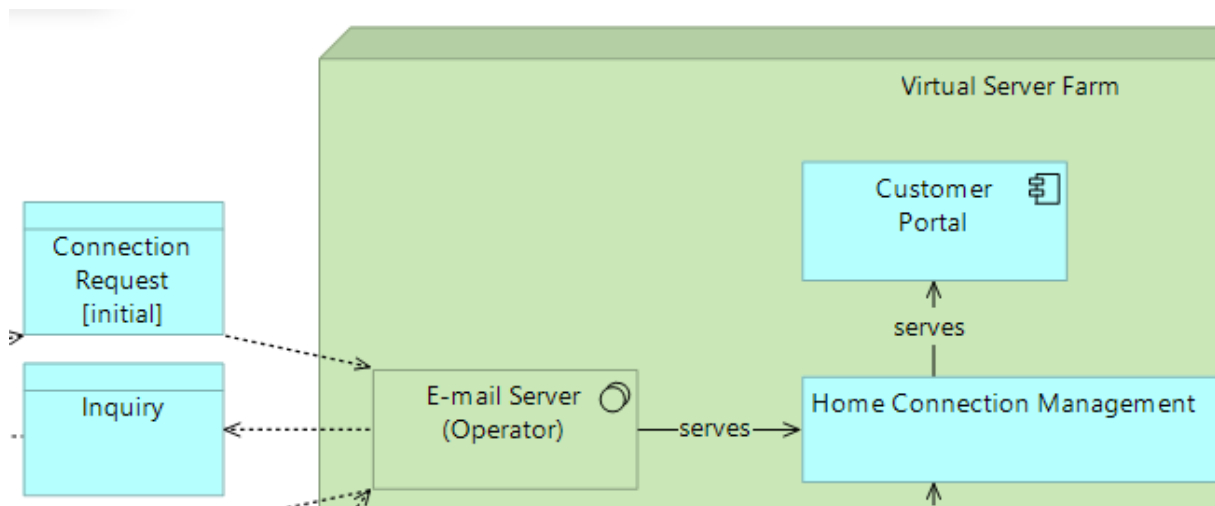
- The listed elements are described in the [ArchiMate® 3.1 Specification](#).
- All SpecIF statement class terms are defined in the [Vocabulary](#).
- For all entities in the left column the namespace 'archimate:' is used; the XML header defines
`xsi:schemaLocation="http://www.opengroup.org/xsd/archimate/3.0/
http://www.opengroup.org/xsd/archimate/3.1/archimate3_Diagram.xsd".`
- The original model element type is stored in a property named *dcterms:type*.

9.1.3 Example

ArchiMate® 3.1 defines numerous viewpoints, 25 in total. Two typical ones are chosen to demonstrate an ArchiMate® to SpecIF transformation.

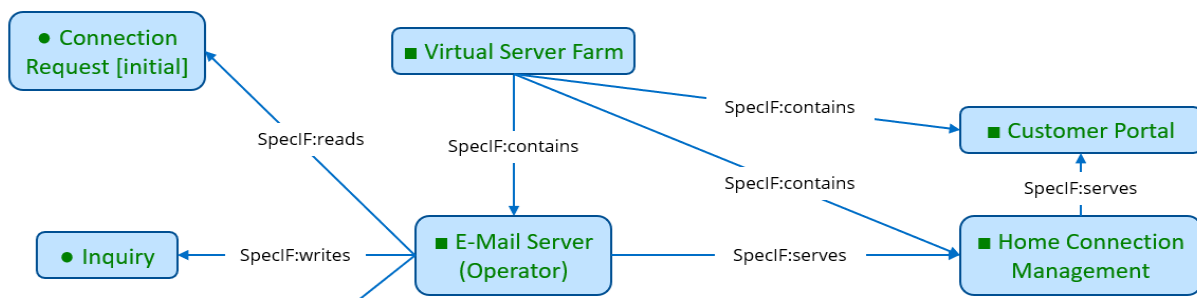
9.1.3.1 Layered Viewpoint

The following clipping from ArchiMate® Layered Viewpoint represents two data objects, one system service, two application components and a node:



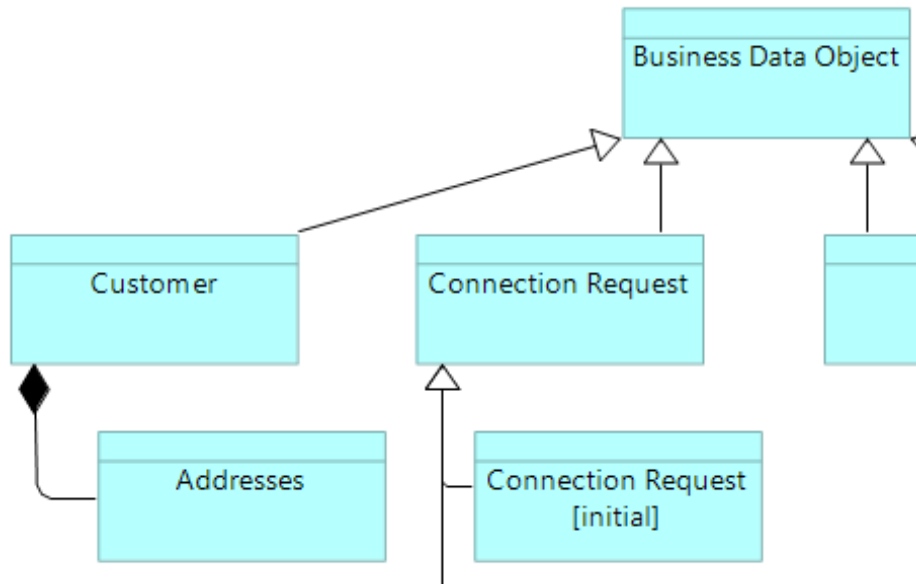
ArchiMate® Layered Viewpoint Clipping

The following SpecIF graph expresses the same:



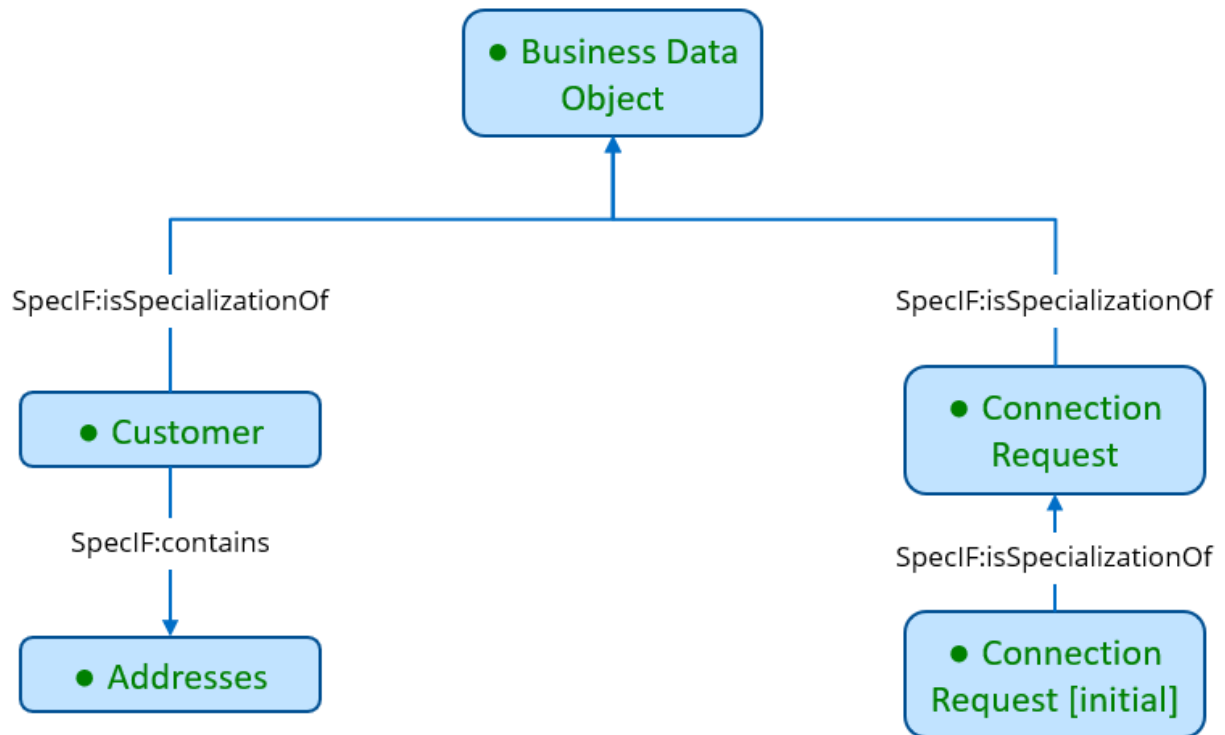
SpecIF from ArchiMate® Layered Viewpoint Clipping

The following clipping from ArchiMate® Information Structure Viewpoint represents five data objects similar to a UML Class Diagram:



ArchiMate® Information Structure Viewpoint Clipping

The following SpecIF graph expresses the same:



SpecIF from ArchiMate® Information Structure Viewpoint Clipping

9.1.3.3 Full Example

The full example can be inspected, here:

- [Telephone Connection Request \(Open-Exchange XML\)](#)
- [Telephone Connection Request \(specif\)](#)
- [Telephone Connection Request \(SpecIF-Viewer\)](#)

9.1.4 Transformation Code

Here you may look at the current code of the [ArchiMate® to SpecIF transformation](#).

10 SpecIF Model Integration Guide for BPMN

10.1 BPMN to SpecIF mapping

For introduction to the Business Process Model and Notation (BPMN), see the [BPMN 2.0 Symbol Reference](#), for example.

10.1.1 Resources

BPMN-XML	SpecIF
collaboration	SpecIF:Diagram
process1	<i>tbd</i>
participant2, laneSet, lane3, task, manualTask, userTask, scriptTask, serviceTask, sendTask, receiveTask, callActivity, transaction, subProcess, businessRuleTask, forking and joining parallelGateway, joining exclusiveGateway, joining inclusiveGateway	FMC:Actor
forking exclusiveGateway, forking inclusiveGateway, forking eventBasedGateway	FMC:Actor plus FMC:Event per outgoing sequenceFlow
dataObjectReference4, dataStoreReference4, messageFlow5	FMC:State
startEvent, timerStartEvent, messageStartEvent, intermediateEvent, messageThrowEvent, intermediateThrowEvent, intermediateCatchEvent, intermediateTimerCatchEvent, intermediateMessageCatchEvent, boundaryEvent, timerBoundaryEvent, messageBoundaryEvent, endEvent	FMC:Event
group6	SpecIF:Collection

Where:

- For all entities in the left column the namespace ‘bpmn:’ is used; the XML header defines `targetNamespace="http://bpmn.io/schema/bpmn"`.
- All SpecIF resource class terms are defined in the [Vocabulary](#).
- The original model element type is stored in a property named `dcterms:type`.

Comments:

1. The participants are transformed, but not the processes, for the following reasons:
 - To our experience with different tools, there is no process without participant, but there can be participants without a process.
 - Participants are source or target of messageFlows, but not the referenced processes.
 - Thus, the role and need of a process in BPMN as implemented by the tools we know is unclear.
2. A participant is also called a ‘pool’.
3. A lane is considered a responsible person or role, thus an FMC:Actor.
4. Interestingly enough, in BPMN the name and other information are properties of `dataObjectReference` resp. `dataStoreReference` (rather than `dataObject` or `dataStore`).

Also the associations point to the references. Therefore, the references are transformed and the `dcterms:type` is anyways set to *dataObject* resp. *dataStore*.

5. A `messageFlow` between different processes is transformed to a *dataObject* with *SpecIF:writes* and *SpecIF:reads* statements connecting to the sending resp. receiving process steps or events.
6. Currently a *group* is not represented as a SpecIF model element, because some (or even all) widely used BPMN modelers do not indicate explicitly in their BPMN-XML export, which activities or other are contained. Thus, the semantics of a group are not easily derived. The geometric coordinates of the diagram can be analyzed to identify contained elements of a group, of course: This is a development task to do.

10.1.2 Statements

At present, the following statements are derived from BPMN diagrams, where the *statement terms (predicates)* are highlighted in *italics*:

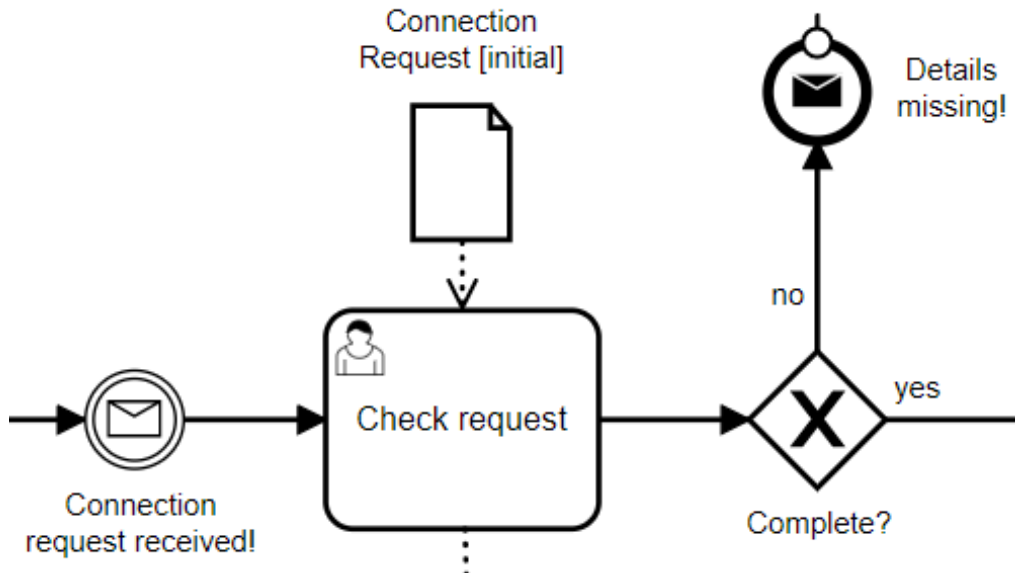
BPMN-XML		SpecIF	Comment
<i>Appearance on diagram</i>	diagram <i>SpecIF:shows</i> model-element	SpecIF:shows	
<i>Graphical Containment</i>	process <i>SpecIF:contains</i> lane	SpecIF:contains	
<i>Graphical Containment</i>	lane <i>SpecIF:contains</i> activity or event	SpecIF:contains	
<i>dataInputAssociation</i>	activity <i>SpecIF:reads</i> data	SpecIF:reads	
<i>dataOutputAssociation</i>	activity <i>SpecIF:writes</i> data	SpecIF:writes	
<i>sequenceFlow</i>	activity <i>SpecIF:precedes</i> activity	SpecIF:precedes	
<i>sequenceFlow</i> ('outgoing' with respect to the event)	event <i>SpecIF:precedes</i> activity	SpecIF:precedes	
<i>sequenceFlow</i> ('incoming' with respect to the event)	activity <i>SpecIF:precedes</i> event	SpecIF:precedes	
<i>association</i>	annotation <i>SpecIF:refersTo</i> model-element	SpecIF:refersTo	

Where:

- For all entities in the left column the namespace 'bpmn:' is used; the XML header defines `targetNamespace="http://bpmn.io/schema/bpmn"`.
- 'model-element' is one of ['FMC:Actor', 'FMC:State', 'FMC:Event']
- 'activity' is one of [task, manualTask, userTask, scriptTask, serviceTask, sendTask, receiveTask, callActivity, transaction, subProcess], thus a FMC:Actor.
- 'data' is one of [dataObjectReference, dataStoreReference], thus a FMC:State.
- 'event' is one of [startEvent, intermediateThrowEvent, intermediateCatchEvent, boundaryEvent, endEvent], thus a FMC:Event.
- All SpecIF statement class terms are defined in the [Vocabulary](#).
- The original model element type is stored in a property named *dcterms:type*.

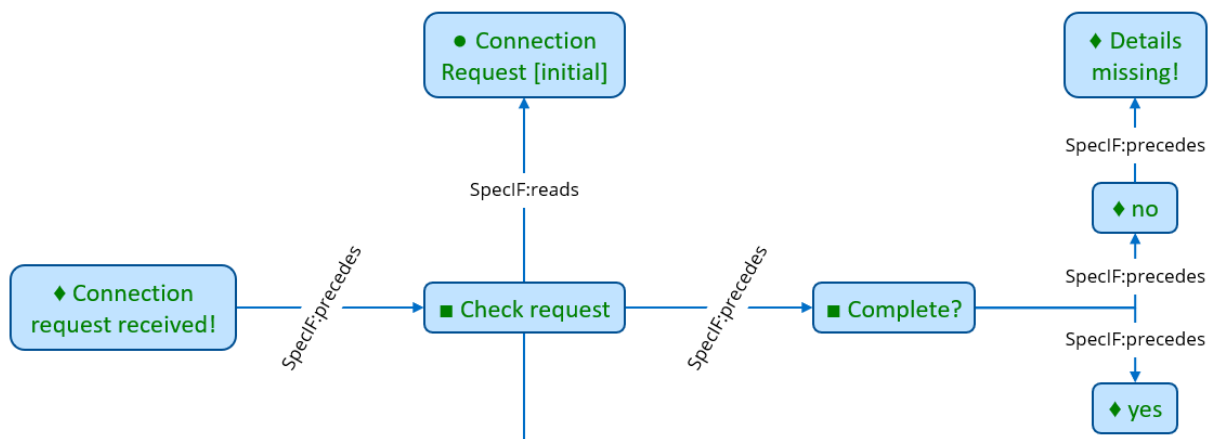
10.1.3 Example

The following clipping from BPMN-XML represents an event, an activity, a sequenceFlow and a forking exclusive gateway:



BPMN Clipping

The following SpecIF graph expresses the same:



SpecIF from BPMN Clipping

The full example can be inspected, here:

- [Telephone Connection Request \(BPMN-XML\)](#)
- [Telephone Connection Request \(specif\)](#)
- [Telephone Connection Request \(SpecIF-Viewer\)](#)

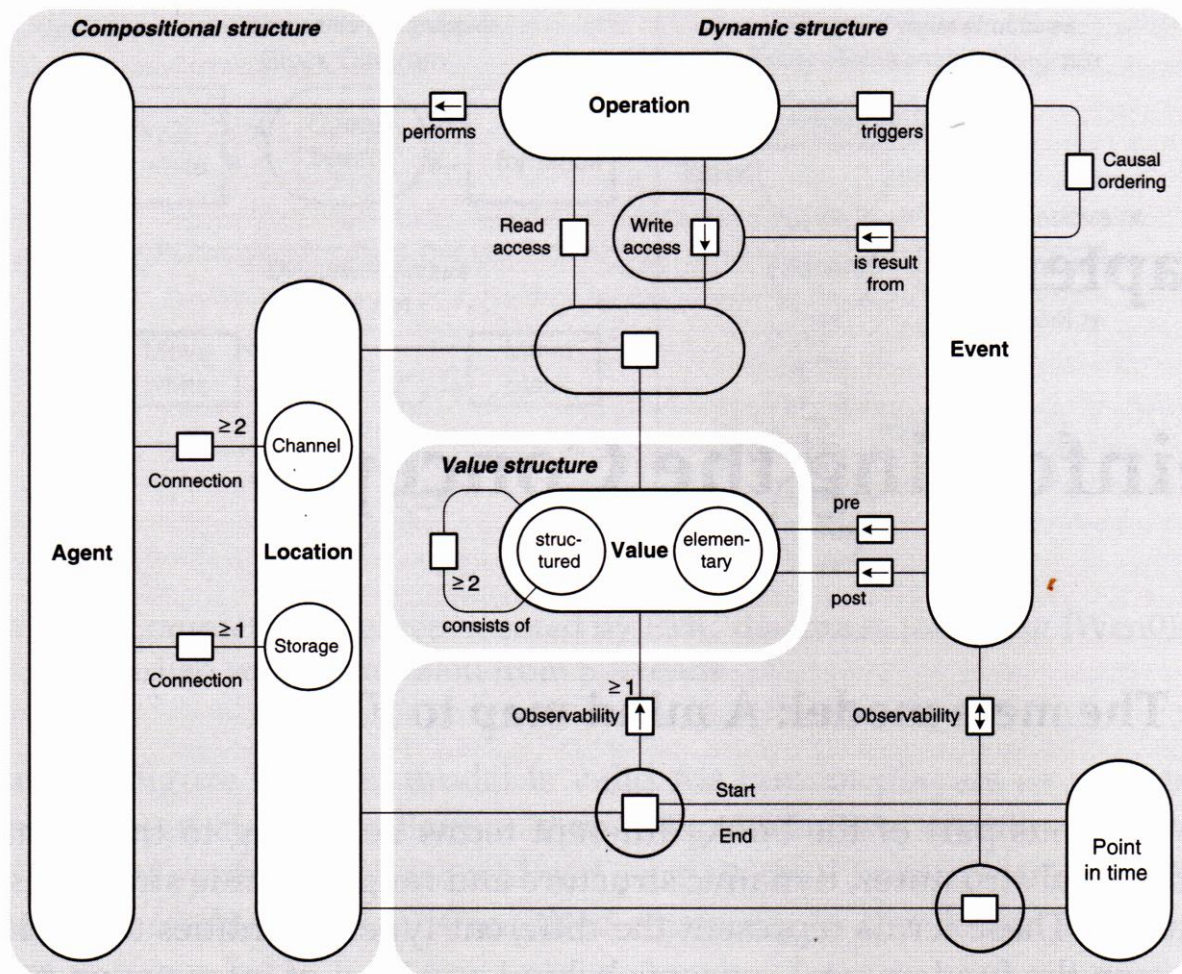
10.1.4 Transformation Code

Here you may look at the current code of the [BPMN to SpecIF transformation](#).

11 SpecIF Model-Integration Guide for FMC

11.1 FMC to SpecIF Mapping

For introduction to the Fundamental Modeling Concepts (FMC), see the [FMC Overview](#). The FMC Metamodel is given in [Knöpfel, Gröne, Tabeing: Fundamental Modeling Concepts - Effective Communication of IT-Systems](#).



FMC Metamodel

However, this Metamodel seems to be rather theoretic and it is perhaps not complete for practical purposes. When analyzing, among others we have the following questions:

- Why are some relations named as nouns, others as verbs (e.g. "*Observability*" vs. "*triggers*").

- Why is *Write access* an unambiguous mapping in one direction (as denoted by the arrow) and why not *Read access*?
- Should the relation between *Location* and *Value* be named *has*?
- Can *Agent*, *Operation*, *Storage* and perhaps other entities be hierarchically nested?

There is no standard serialization defined for FMC. As to our knowledge there is only one modeling tool for FMC, namely [ARCWAY Cockpit](#). We will show in the following the mapping of the entities and relationships realized in this tool.

11.1.1 Resources

ARCWAY Cockpit	SpecIF
Plan	SpecIF:Diagram
Function, Agent, Operation	FMC:Actor
Information, Channel, Location, Storage, Value	FMC:State
Event, Point in time	FMC:Event
	SpecIF:Collection

Where:

- All resource class terms are defined in the [Vocabulary](#).
- The original model element type is stored in a property named *dterms:type*.

11.1.2 Statements

ARCWAY Cockpit		SpecIF	Comment
Occurrence	A model-element occurs on a plan	SpecIF:shows	Inverted statement
Containment	A model-element contains a model-element	SpecIF:contains	
Reading	An actor reads a state	SpecIF:reads	
Writing	An actor writes a state	SpecIF:writes	
Modification	An actor writes and reads a state	SpecIF:stores	
Influence	A state influences a state (via an unnamed function=actor)	<i>tbd</i>	Rarely used, if ever
Relation	A state influences and is reversely influenced by a state (via an unnamed function=actor)	<i>tbd</i>	Rarely used, if ever
Receiving	An actor receives messages from an actor (through an unnamed channel=state)	SpecIF:receivesFrom	
Sending	An actor sends messages to an actor (through an unnamed channel=state)	SpecIF:sendsTo	
Communication	An actor sends to and receives messages from an actor (through an unnamed channel=state)	SpecIF:communicates	
Succession	A model-element succeeds a model-element	SpecIF:precedes	Inverted statement
Precedence	A model-element precedes a model-element	SpecIF:precedes	

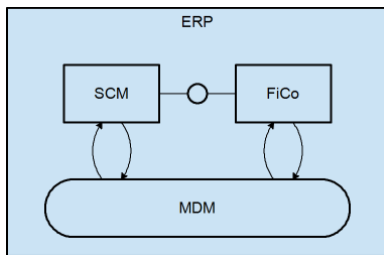
Where:

- ‘model-element’ is one of [‘FMC:Actor’, ‘FMC:State’, ‘FMC:Event’].
- All statement class terms are defined in the [Vocabulary](#).
- The original model element type is stored in a property named *dcterms:type*.

11.1.3 Example

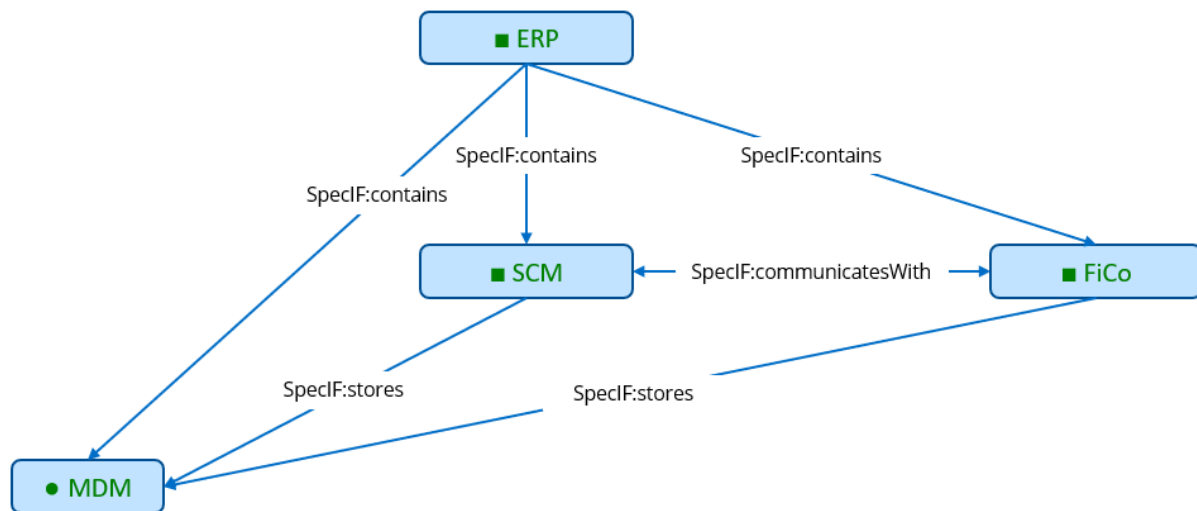
11.1.3.1 FMC Block Diagram

The following example of a FMC block diagram represents three application components and a data object:



FMC Block Diagram

The following SpecIF graph expresses the same:



SpecIF from FMC Block Diagram

11.1.4 Transformation Code

Here you may take a look at the current code of the [FMC Export from ARCWAY Cockpit to SpecIF](#).

12 SpecIF Model Integration Guide for UML and SysML

To semantically map SysML to SpecIF we map the UML and include the elements and concepts of the UML-profile mechanism. All SysML 1.x elements are profiled UML elements with stereotypes and tagged values. So it is easy to map these elements using the UML metaclasses and the stereotypes. SpecIF defines the properties `dterms:type` for defining a UML-metamodel reference and the property `UML:Stereotype` to define a stereotype. So SysML elements are profiled UML elements.

12.1 UML/SysML to SpecIF Mapping

The mapping from UML/SysML to SpecIF is currently still discussed in a working group of the GfSE. At the moment there is a sketch for a mapping of UML and SysML to SpecIF elements. This mapping does not cover the complete UML/SysML today and some parts are still in discussion. It consists of a resource-, property- and statement-mapping and only contains elements that are often used in activity, block definition and other popular diagrams used in Systems and Software Engineering. The mapping is shown in the following tables.

The general idea is to map a high number of UML/SysML elements to a small number of elements from SpecIF. The most elements are the elements of the *SpecIF Integration Model*. This model core consists of the fundamental modeling elements Actor, State and Event and the two elements SpecIF:Diagram, UML:Package and SpecIF:Collection. Mapped elements in the mapping tables should have a matching semantic meaning, e.g. a SysML Action is an active element. Regarding to the SpecIF class definitions, a FMC:Actor represents an active element. Since these definitions match both elements can be added to the mapping table.

To define the values for the `dterms:type` property, the UML metamodel in version 2.5.1 (OMG document number: formal/2017-12-05) is used as reference.

12.1.1 Mapping of UML Profiles

By defining a so called UML-Profile, it is possible to define new element types from existing UML elements with additional meanings or additional properties. The elements defined by a profile extension are marked by a special term, called a Stereotype. SysML in version 1.x is a modeling language that is defined as an UML-profile as well. So in the mapping tables below

one column is showing the Stereotype if necessary. When an UML-element has no stereotype or a stereotype that is not yet covered by the mapping tables, the element should be mapped in the same way as the UML element without a stereotype (UML native element type)!

For example the UML element of the UML-type *Interface* with Stereotype *myStereotype* is mapped in the same way as an Interface **without** a stereotype (to a FMC:State with dterms:type set to OMG:UML:2.5.1:Interface), but the SpecIF property *UML:Stereotype* must then be set to *myStereotype*.

12.1.2 Resource mapping tables

12.1.2.1 Package mappings

UML-Metaclass	UML:Stereotype	SpecIF Resource Class	dterms:type	Remark
Model	-	UML:Package	OMG:UML:2.5.1:Model	A root model node (root package of a model repository).
Package	-	UML:Package	OMG:UML:2.5.1:Package	A model package

12.1.2.2 Diagram mappings

UML-Metaclass	UML:Stereotype	SpecIF Resource Class	dterms:type	Remark
ClassDiagram	-	SpecIF:Diagram	OMG:UML:2.5.1:ClassDiagram	A UML class diagram
CompositeStructureDiagram	-	SpecIF:Diagram	OMG:UML:2.5.1:CompositeStructureDiagram	A UML composite structure diagram
ComponentDiagram	-	SpecIF:Diagram	OMG:UML:2.5.1:ComponentDiagram	A UML component diagram
DeploymentDiagram	-	SpecIF:Diagram	OMG:UML:2.5.1:DeploymentDiagram	A UML deployment diagram
ObjectDiagram	-	SpecIF:Diagram	OMG:UML:2.5.1:ObjectDiagram	A UML object diagram
PackageDiagram	-	SpecIF:Diagram	OMG:UML:2.5.1:PackageDiagram	A UML/SysML package diagram
ProfileDiagram	-	SpecIF:Diagram	OMG:UML:2.5.1:ProfileDiagram	A UML/SysML profile diagram
ActivityDiagram	-	SpecIF:Diagram	OMG:UML:2.5.1:ActivityDiagram	A UML/SysML activity diagram



UML-Metaclass	UML:Stereotype	SpecIF Resource Class	dcterms:type	Remark
SequenceDiagram	-	SpecIF:Diagram	OMG:UML:2.5.1:SequenceDiagram	A UML/SysML sequence diagram
CommunicationDiagram	-	SpecIF:Diagram	OMG:UML:2.5.1:CommunicationDiagram	A UML communication diagram
InteractionOverviewDiagram	-	SpecIF:Diagram	OMG:UML:2.5.1:InteractionOverviewDiagram	A UML interaction overview diagram
TimingDiagram	-	SpecIF:Diagram	OMG:UML:2.5.1:TimingDiagram	A UML timing diagram
UseCaseDiagram	-	SpecIF:Diagram	OMG:UML:2.5.1:UseCaseDiagram	A UML/SysML use case diagram
StateMachineDiagram	-	SpecIF:Diagram	OMG:UML:2.5.1:StateMachineDiagram	A UML/SysML state machine diagram
SysML-RequirementDiagram	Requirement	SpecIF:Diagram	OMG:UML:2.5.1:ClassDiagram	A SysML Requirement Diagram
SysML-BlockDefinitionDiagram	BlockDefinition	SpecIF:Diagram	OMG:UML:2.5.1:ClassDiagram	A SysML Block Definition Diagram
SysML-InternalBlockDiagram	InternalBlock	SpecIF:Diagram	OMG:UML:2.5.1:ObjectDiagram	A SysML Internal Block Diagram
SysML-ParametricDiagram	Parametric	SpecIF:Diagram	OMG:UML:2.5.1:ObjectDiagram	A SysML Parametric Constraint Diagram
FMC4SE FMC Diagram	FMC4SE	SpecIF:Diagram	OMG:UML:2.5.1:ObjectDiagram	A FMC4SE FMC Diagram

12.1.2.3 State mappings (passive elements)

UML-Metaclass	UML:Stereotype	SpecIF Resource Class	dcterms:type	Remark
Class	-	FMC:State	OMG:UML:2.5.1:Class	We know that a class can have behavioral parts. But a class is mostly used to define data structures, so it was decided to map it to the passive elements.
Interface	-	FMC:State	OMG:UML:2.5.1:Interface	A UML interface element.
Enumeration	-	FMC:State	OMG:UML:2.5.1:Enumeration	A UML enumeration element.



UML-Metaclass	UML:Stereotype	SpecIF Resource Class	dcterms:type	Remark
State	-	FMC:State	OMG:UML:2.5.1:State	A state in a state machine/state chart.
Port	-	FMC:Actor (+ FMC:State)	OMG:UML:2.5.1:Port	The Port is mapped to an FMC:Actor and connected to a FMC:State with <code>rdf:type == typeOf(Port)</code> by using a <code>SpecIF:reads/writes/precedes</code> statement.
Attribute	-	FMC:State	OMG:UML:2.5.1:Attribute	An attribute of a class. Typically connected by <i>SpecIF:contains</i> from the class resource.
Object	-	FMC:State	OMG:UML:2.5.1:Object	An UML object.
TaggedValue	-	FMC:State	OMG:UML:2.5.1:TaggedValue	A tagged value. Very important for profiles like SysML.
ObjectRunState	-	FMC:State	OMG:UML:2.5.1:RunState	An object run state with a concrete value.
Constraint	-	FMC:State	OMG:UML:2.5.1:Constraint	A model constraint.
Parameter	-	FMC:State	OMG:UML:2.5.1:Parameter	An operation parameter.
Class	block	FMC:State	OMG:UML:2.5.1:Class	A SysML Block.
Class	valueType	FMC:State	OMG:UML:2.5.1:Class	A SysML ValueType.
Class	constraint	FMC:State	OMG:UML:2.5.1:Class	A SysML ConstraintBlock.

12.1.2.4 Actor mappings (active elements)

UML-Metaclass	UML:Stereotype	SpecIF Resource Class	dcterms:type	Remark
Component	-	FMC:Actor	OMG:UML:2.5.1:Component	A UML Component.
Property	-	FMC:Actor	OMG:UML:2.5.1:Property	A UML/SysML Property.
SysML-ConstraintProperty	constraintProperty	FMC:Actor	OMG:UML:2.5.1:Property	A SysML ConstraintProperty (Instance of a ConstraintBlock).
Activity		FMC:Actor	OMG:UML:2.5.1:Activity	An activity definition.
Action	-	FMC:Actor	OMG:UML:2.5.1:Action	A Action is typically an instance of an Activity.
CallBehaviorAction	-	FMC:Actor	OMG:UML:2.5.1:CallBehaviorAction	
UseCase	-	FMC:Actor	OMG:UML:2.5.1:UseCase	A UML/SysML use case.
InputPin	-	FMC:Actor	OMG:UML:2.5.1:InputPin	
OutputPin	-	FMC:Actor	OMG:UML:2.5.1:OutputPin	
Operation	-	FMC:Actor	OMG:UML:2.5.1:Operation	A class or State operation.
StateMachine	-	FMC:Actor	OMG:UML:2.5.1:StateMachine	A statechart state machine.

UML-Metaclass	UML:Stereotype	SpecIF Resource Class	dcterms:type	Remark
Actor	-	FMC:Actor	OMG:UML:2.5.1:Actor	A human actor.
DecisionNode	-	FMC:Actor + FMC:Event (+ SpecIF:signals + SpecIF:triggers)	OMG:UML:2.5.1:DecisionNode	Incoming edge: SpecIF:precedes; Outgoing edge: SpecIF:signals an FMC:Event and the event SpecIF:triggers the next FMC:Actor.
MergeNode	-	FMC:Actor	OMG:UML:2.5.1:MegeNode	As statement SpecIF:precedes is used.
ForkNode	-	FMC:Actor	OMG:UML:2.5.1:ForkNode	
JoinNode	-	FMC:Actor	OMG:UML:2.5.1:JoinNode	
ActivityPartition	-	FMC:Actor (+ SpecIF:contains)	OMG:UML:2.5.1:ActivityPartition	
SendSignalAction	-	FMC:Actor (+ SpecIF:precedes with dcterms:type = SpecIF:signals)	OMG:UML:2.5.1:SendSignalAction	
AcceptEventAction	-	FMC:Actor (+ SpecIF:precedes with dcterms:type = SpecIF:triggers)	OMG:UML:2.5.1:AcceptEventActio n	

12.1.2.5 Event mappings

UML-Metaclass	UML:Stereotype	SpecIF Resource Class	dcterms:type	Remark
TimeEvent	-	FMC:Event	OMG:UML:2.5.1:TimeEvent	
Signal	-	FMC:Event	OMG:UML:2.5.1:Signal	
InitialNode	-	FMC:Event	OMG:UML:2.5.1:InitialNode	
ActivityFinalNode	-	FMC:Event	OMG:UML:2.5.1:ActivityFinal	
FlowFinalNode	-	FMC:Event	OMG:UML:2.5.1:FlowFinal	

12.1.2.6 Requirement mappings

UML-Metaclass	UML:Stereotype	SpecIF Resource Class	dcterms:type	Remark
Class	requirement	IREB:Requirement	-	A SysML requirement model element. Also map tool-specific requirement representations (e.g. EA-Requirement-Elements) to this resource class.

12.1.2.7 Other resource mappings

UML-Metaclass	UML:Stereotype	SpecIF Resource Class	dcterms:type	Remark
Comment	-	SpecIF:Comment	-	A UML comment (note) element.
UseCase Subject	-	SpecIF:Collection	-	UML-Boundary elements used in UML/SysML use case diagrams.

12.1.3 Statement mappings

UML Metaclass	Stereotype	SpecIF Statement Class	dterms:type	Remark
ObjectFlow	-	FMC:State + SpecIF:reads/writes + SpecIF:precedes	OMG:UML:2.5.1:ObjectFlow	FMC:State + SpecIF:reads/writes to transfer the Object, additionally a control flow to trigger the reading actor
ControlFlow	-	SpecIF:precedes	OMG:UML:2.5.1:ControlFlow	The connection type (precedes with dterms:ttype = triggers or signals) depends on the types of the connected elements
Transition	-	FMC:Event + FMC:Actor + SpecIF:precedes + SpecIF:reads/writes		Used to interconnect states. See state mapping example for details.
Connector (w/o direction)	access type	SpecIF:stores	OMG:UML:2.5.1:Connector	Used in FMC4SE compositional structure modeling (—)
Connector (Unidirectional)	access type	SpecIF:writes/SpeIF:reads	OMG:UML:2.5.1:Connector	Used in FMC4SE compositional structure modeling (→)
Connector (Bi-Directional)	access type	SpecIF:stores	OMG:UML:2.5.1:Connector	Used in FMC4SE compositional structure modeling (<→)
Composition	-	SpecIF:contains	OMG:UML:2.5.1:CompositeAggregation	UML/SysML composition (black diamond)
Aggregation	-	SpecIF:contains	OMG:UML:2.5.1:Aggregation	UML/SysML aggregation (white diamond)
Association	-	SpecIF:contains (one or multiple)	OMG:UML:2.5.1:Association	UML/SysML association defining an attribute
Association	-	SpecIF:isAssociatedWith	OMG:UML:2.5.1:Association	UML/SysML association expressing a logical association
Generalization	-	SpecIF:isSpecializationOf	OMG:UML:2.5.1:Generalization	UML/SysML generalization ('inheritance relation')
Dependency	-	SpecIF:dependsOn	OMG:UML:2.5.1:Dependency	UML/SysML dependency



UML Metaclass	Stereotype	SpecIF Statement Class	dcterms:type	Remark
Dependency	satisfy	oslc_rm:satisfies	OMG:UML:2.5.1:Dependency	SysML satisfy connection
Dependency	verify	SysML:verifies	OMG:UML:2.5.1:Dependency	SysML verify connection
Dependency	allocate	SpecIF:allocates	OMG:UML:2.5.1:Dependency	SysML allocation connection
Deployment	deploy	SpecIF:allocates	OMG:UML:2.5.1:Deployment	UML deployment connection
Extension	-	UML:Extends	OMG:UML:2.5.1:Extension	UML Profile extension relation. The subject (stereotype) extends the object (metaclass).
OwnedComment	-	SpecIF:refersTo	OMG:UML:2.5.1:OwnedComment	Connection between a Comment and an UML element (also known as note link).
Trigger	-	SpecIF:triggers		Triggers an AcceptEventAction
Signal	-	SpecIF:signals		Signals a SignalEvent
Classifier of a model element/Resource	-	rdf:type		The object is the classifier/type of the subject.
behavior is implemented by	-	UML:BehaviorReference?		The subject behavior is implemented by the object (a piece of source code)

12.1.4 Property mappings

UML-Name	SpecIF-Property Class	Remark
NamedElement.name	dcterms:title	The name of an UML element.
Comment	dcterms:description	The model element descriptional text.
NamedElement.visibility	SpecIF:VisibilityKind	e.g. Public, Private, Protected, Package
?	SpecIF:Status	The element status value.
Stereotype	UML:Stereotype	The Stereotype of a model element
typeOf(metaclass)	dcterms:type	e.g. OMG:UML:2.5.1:Constraint. To avoid define a SpecIF Resource type for each UML element, a type attribute is used instead of metaclass inheritance. Have a look at the dcterms:type column in the mapping tables for details.
VALUES (TaggedValue.Value / Attribute.Default / Constraint.Notes)	rdf:value	An (initial) value of an attribute, tagged value, object run state etc.

UML-Name	SpecIF-Property Class	Remark
alias	SpecIF:Alias	An alias for a model element.

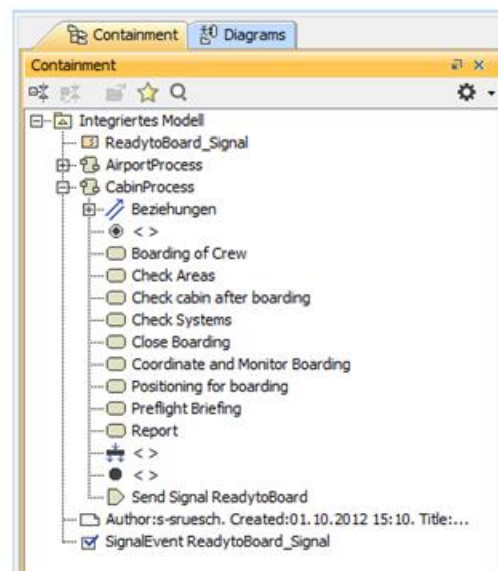
12.2 Mapping of the model structure

Beside the mapping of model elements and connectors from UML to resources and statements in SpecIF, UML/SysML tools provide a tree structure for bringing structure to the model(-repository) itself. This model tree structure shall be mapped to SpecIF using SpecIF hierarchy/node elements. So after the transformation to SpecIF this structure is provided by SpecIF hierarchy.

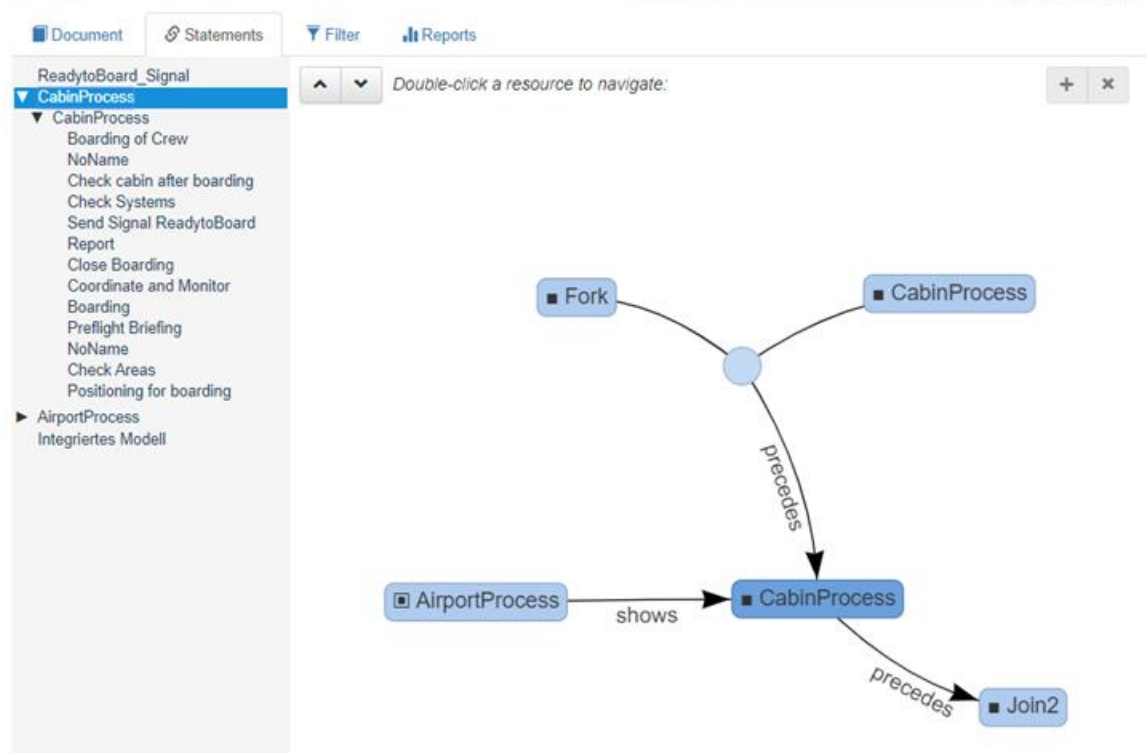
Different structure levels are defined by hierarchy elements and nodes. E.g. an activity can be a hierarchy element. A node list can be added to this hierarchy element to create a structure. One hierarchy element is always linked to one model element. In the figure below the model structure is shown in a Cameo Systems Modeler containment tree and in a SpecIF-Viewer. A SpecIF-Viewer is offering the possibility to read SpecIF files and create an overview. The model structure is shown on the left side in the hierarchy tree. Information about model elements are provided in the “document” section and relations to other model elements are provided in the “statements” section.



Structure Mapping



Integrated Model



Structure Mapping

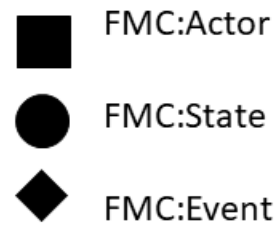
In the UML tool Sparx Enterprise Architect the model structure tree is shown similar to the Cameo containment tree in a “Project Browser” window. When model data is transformed to SpecIF this tree structure shall be mapped one to one to a SpecIF hierarchy so that a data and structure exchange is possible between different UML tools.







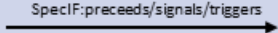






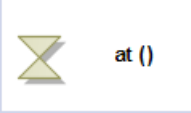



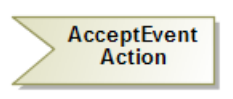

12.3 Examples

12.3.1 Examples for UML/SysML mapping and transformation of activity diagrams

In SpecIF model elements are represented as FMC:Actor (square), FMC:State (circle) and FMC:Event (diamond). Elements in SpecIF are named resources and relations between these resources are named statements (arrows). The following figure shows the element representation in Cameo Systems Modeler and SpecIF. The notation that is shown in the figure is also used for the following examples.

Element - Representation

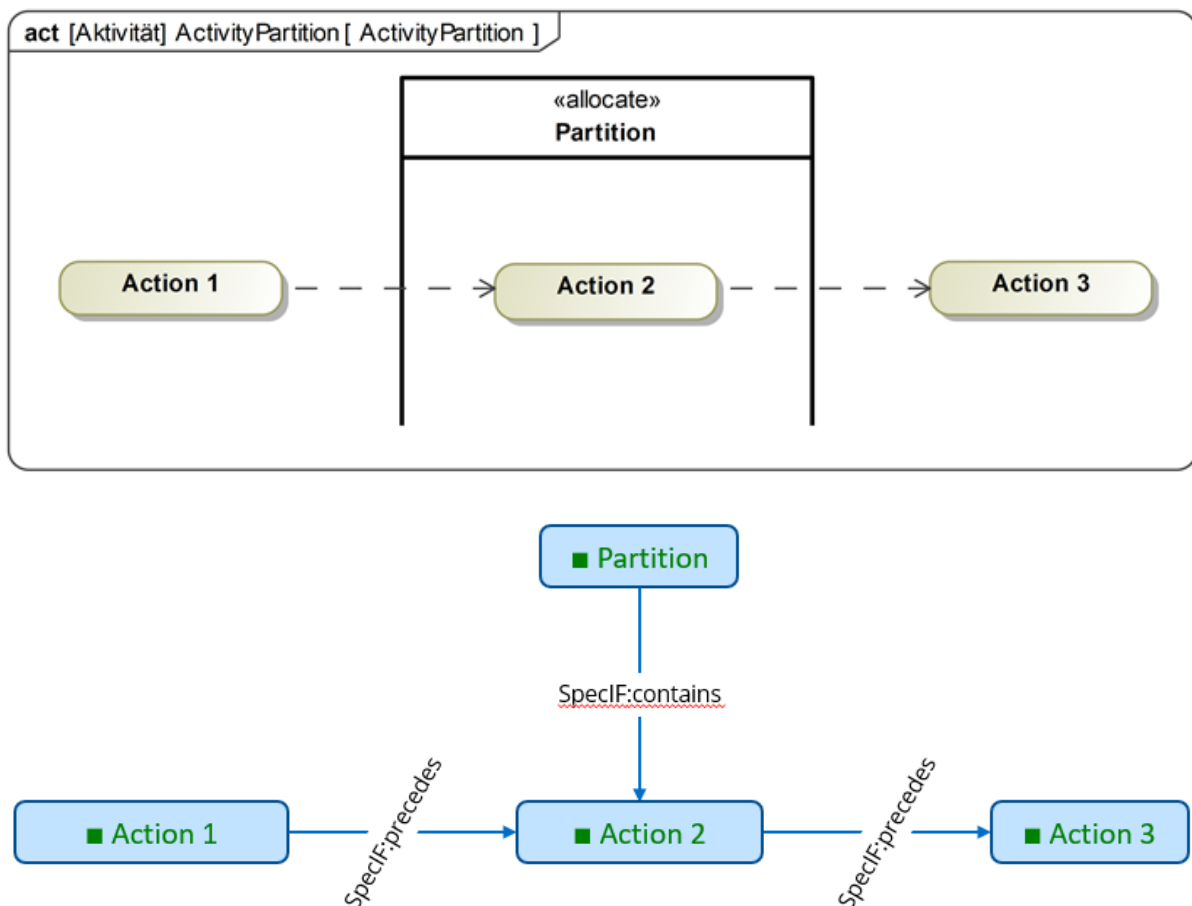


UML Name	Representation in Cameo	Representation in SpecIF
uml:CallBehaviorAction		
uml:ControlFlow		SpecIF:preceeds/signals/triggers 
uml:ObjectFlow		SpecIF:writes  SpecIF:reads SpecIF:preceeds/signals/triggers 
uml:OutputPin uml:InputPin		SpecIF:contains 
uml:InitialNode		
uml:ActivityFinalNode		
uml:TimeEvent		
uml:SendSignalAction		
uml:AcceptEventAction		

Mapping SysML Representation

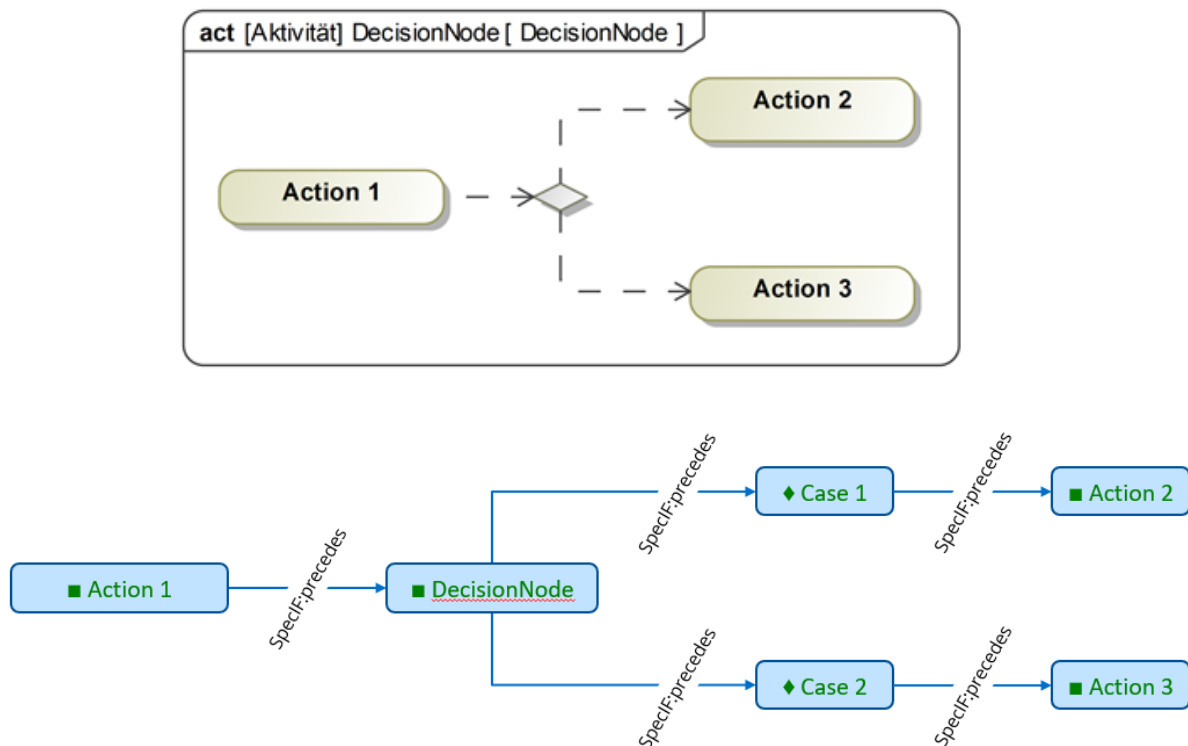
In the next three figures examples for the transformation of activity diagram elements are shown in a small context. The example elements are an ActivityPartition, a DecisionNode and a MergeNode. The first figure shows three Actions, two ControlFlows and a ActivityPartition in a UML or SysML diagram and in the SpecIF notation. Actions are transformed into FMC:Actor resources as well as the activity partition. The graphical containment of *Action 2* in the ActivityPartition is transformed to a statement SpecIF:contains. SpecIF:precedes represents a ControlFlow between two FMC:Actors.

uml:ActivityPartition



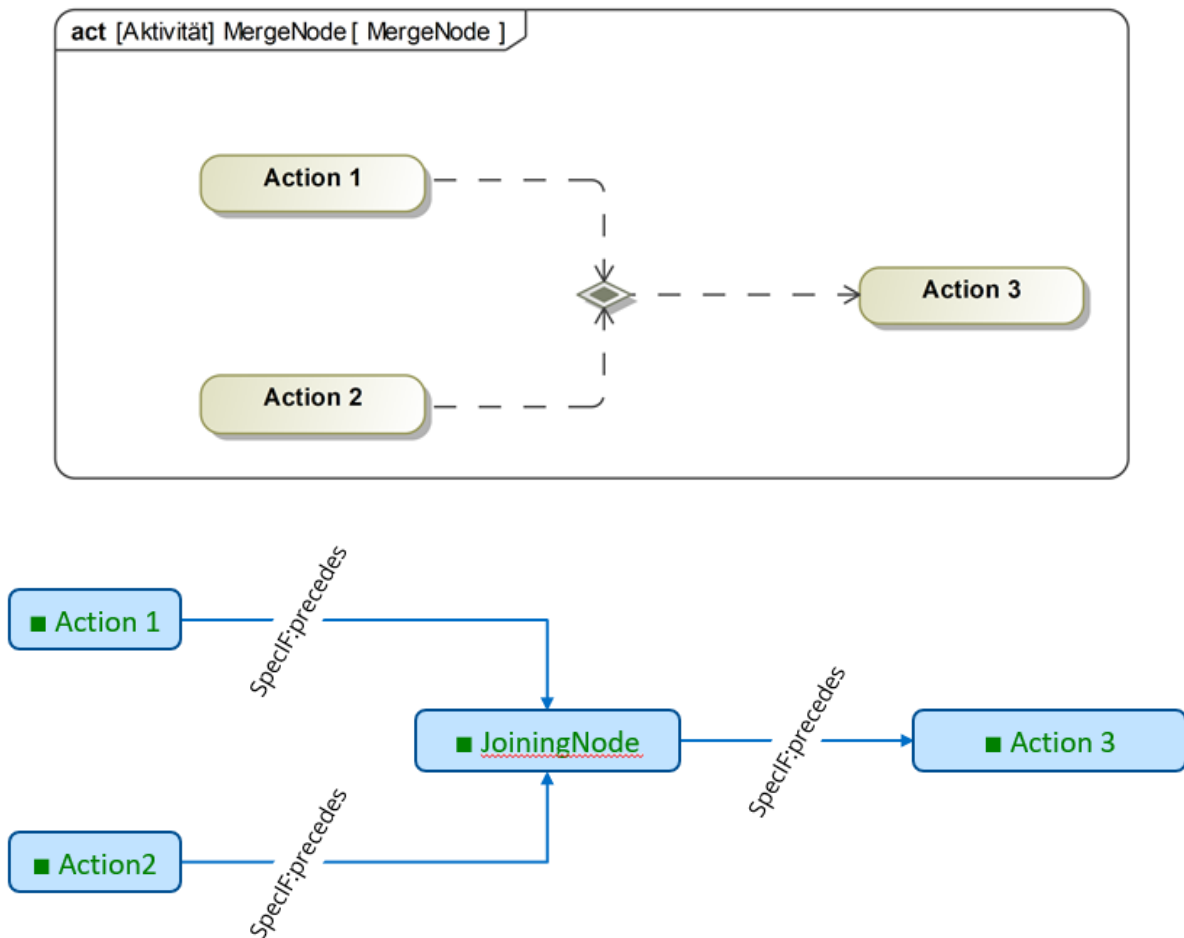
The following figure shows an example for the transformation of a DecisionNode. In the example, the DecisionNode has one incoming Control flow from *Action 1* and two outgoing control flows to *Action 2* and *Action 3*. In SpecIF-Notation, the DecisionNode is represented by a *FMC:Actor* and a *FMC:Event* per decision case. The control flows are again transformed to *SpecIF:precedes*.

uml:DecisionNode



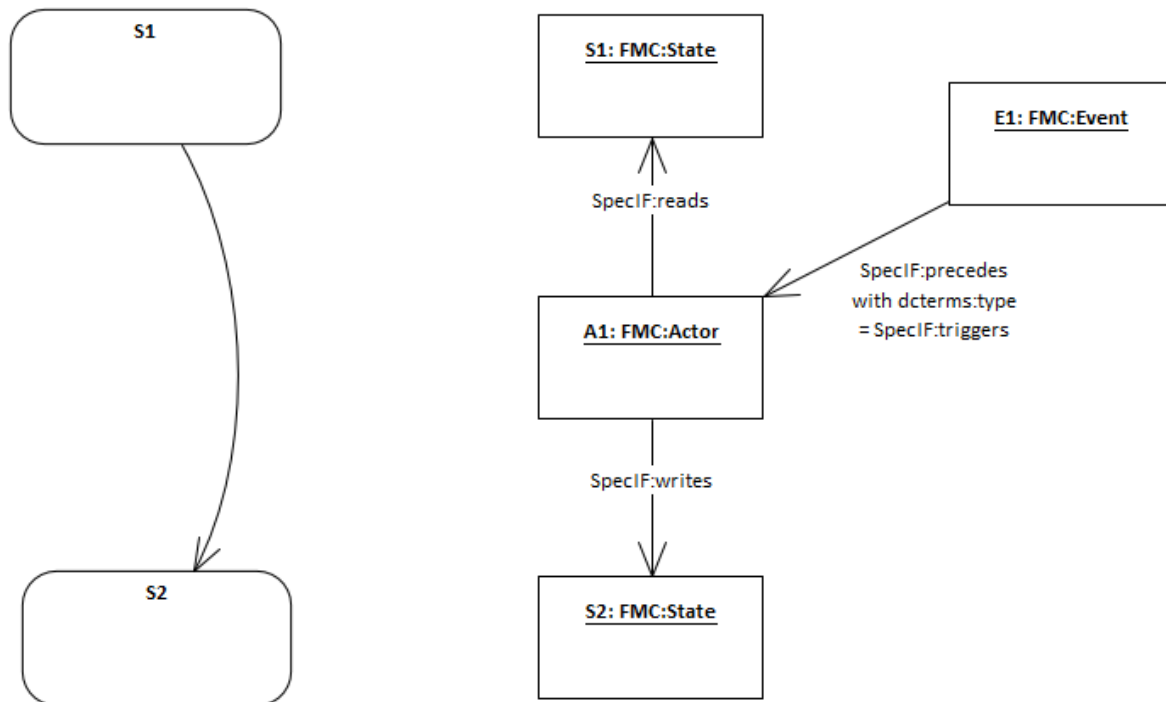
In a MergeNode two incoming activity flows are merged into one outgoing control flow. The JoiningNode is transformed into a FMC:Actor and the control flows to *SpecIF:precedes*.

uml:MergeNode



12.3.2 Example for mapping state charts and state transitions to the SpecIF Integration Model elements

In UML/SysML Statechart diagrams a transition between two states can be modeled in different ways, but result in the same semantic result (same behavior). To bring a statechart to live activities are assigned to states and state transitions. When a state is entered, exited or when the state is active one or more action can be executed to execute some behavior (or code). Also an action can be executed when the state transition is done. A state in a statechart defines for that purpose the entry-, exit- or do-actions. From the semantic point of view it makes no difference if an action is executed when one state is exited as exit-action or as entry-action when the next state is entered, because per definition a state transition should consume no time. So for the semantic integration used by SpecIF, all these semantic equal concepts are mapped to the same schema using the SpecIF integration model elements.



Mapping of a state transition

The diagram above shows an small example how to map the behavior of a state transition to the concepts of FMC semantic modeling used in SpecIF. In the UML/SysML statechart on the left side you gave two states (S1 and S2) connected by a transition. No further actions are assigned in this example diagram. A transition between two states shall be mapped in SpecIF to the two state elements with an additional event and actor element between the two states. You see the equivalent SpecIF elements on the right side of the diagram using the notation of UML object diagrams. The FMC:Actor element between the two states has no behavior in our example, because no behavior is defined in the state chart.

So all transitions are mapped using this approach defined by FMC and Petri-Nets used to define behavior with the SpecIF integration model.

If there are behavioral elements (entry-, exit- or do-actions) or events defined explicitly in the state chart, they are mapped straight forward using multiple FMC:Actor resources with SpecIF:precedes statements or further FMC:Event resources.

13 SpecIF-ReqIF Mapping

Transforming SpecIF to ReqIF is bidirectional. Files can be exported and imported back and forth without losing any data.

Note that there are SpecIF-Attributes which don't have a ReqIF equivalent yet, such as "revisions", "replaces" and "alternativeIds". A way to utilize those attributes is creating an additional ATTRIBUTE-DEFINITION in ReqIF.

13.1 Datatypes

13.1.1 Strings

SpecIF	ReqIF
id	IDENTIFIER
changedAt	LAST-CHANGE
title	LONG-NAME
description	DESC
type: xs:string	REQIF:DATATYPE-DEFINITION-STRING
maxLength	MAX-LENGTH

13.1.2 Boolean

SpecIF	ReqIF
id	IDENTIFIER
title	LONG-NAME
description	DESC
type: xs:boolean	DATATYPE-DEFINITION-BOOLEAN
changedAt	LAST-CHANGE

13.1.3 Byte

SpecIF	ReqIF
id	IDENTIFIER
title	LONG-NAME
description	DESC
type: xs:integer	DATATYPE-DEFINITION-INTEGER
minInclusive	MIN
maxInclusive	MAX
changedAt	LAST-CHANGE

13.1.4 Integer

SpecIF	ReqIF
id	IDENTIFIER
title	LONG-NAME
description	DESC



SpecIF	ReqIF
type: xs:integer	DATATYPE-DEFINITION-INTEGER
minInclusive	MIN
maxInclusive	MAX
changedAt	LAST-CHANGE

13.1.5 Real

SpecIF	ReqIF
id	IDENTIFIER
title	LONG-NAME
description	DESC
changedAt	LAST-CHANGE
type: xs:double	DATATYPE-DEFINITION-REAL
maxInclusive	MAX
minInclusive	MIN
fractionDigits	ACCURACY

13.1.6 Date

SpecIF	ReqIF
id	IDENTIFIER
title	LONG-NAME
description	DESC
changedAt	LAST-CHANGE
type: xs:dateTime	DATATYPE-DEFINITION-DATE

13.1.7 XHTML

SpecIF	ReqIF
id	IDENTIFIER
title	LONG-NAME
description	DESC
changedAt	LAST-CHANGE
type: xs:string; format: "xhtml"	DATATYPE-DEFINITION-XHTML

13.1.8 Enumeration

SpecIF	ReqIF
id	IDENTIFIER
title	LONG-NAME
description	DESC
type: xs:enumeration	DATATYPE-DEFINITION-ENUMERATION
values[]	ENUM-VALUE
changedAt	LAST-CHANGE

Example for values in SpecIF:

```
"enumeration":
  [
    {
      "id": "V-Status-0",
      "value": [
        {
          "text": "SpecIF:LifecycleStatusDeprecated"
        }
      ]
    },
    ...
  ]
```

Example in ReqIF:

```
<ENUM-VALUE IDENTIFIER="V-Status-0" LONG-NAME="SpecIF:LifecycleStatusDeprecated" LAST-CHANGE="2016-05-26T08:59:00+02:00">
  <PROPERTIES>
    <EMBEDDED-VALUE KEY="0" OTHER-CONTENT=""/>
  </PROPERTIES>
</ENUM-VALUE>
```

Redundant data in this example like LAST-CHANGE is deduplicated by SpecIF.

13.2 SpecIF schema attributes

SpecIF has propertyClasses that can be used by all resourceClasses and statementClasses, while in ReqIF they have to be defined anew for each OBJECT-TYPE and RELATION-TYPE.

SpecIF has no equivalent for SPECIFICATION-TYPES, instead a normal resource is used as root.

13.2.1 ResourceClasses

SpecIF	ReqIF
id	IDENTIFIER
title	LONG-NAME
description	DESC
propertyClasses[]	SPEC-ATTRIBUTES
changedAt	LAST-CHANGE

resourceClass example:

in SpecIF:

```
"resourceClasses":
  [
    {
      "id": "RC-Folder",
```



```
"title": "SpecIF:Heading",
"description": [
  {
    "text": "Folders with title and text for chapters or descriptive paragraphs."
  }
],
"revision": "1.1",
"replaces": [],
"icon": "H",
"isHeading": true,
"instantiation": [
  "auto",
  "user"
],
"propertyClasses": [
  {
    "id": "PC-Name",
    "revision": "1.1"
  },
  {
    "id": "PC-Description",
    "revision": "1.1"
  },
  {
    "id": "PC-Type",
    "revision": "1.1"
  }
],
"changedAt": "2016-05-26T08:59:00+02:00"
]
```

in ReqIF:

```
<SPEC-OBJECT-TYPE IDENTIFIER="RC-Folder" LONG-NAME="SpecIF:Heading" DESC="Folders with title and text for chapters or descriptive paragraphs." LAST-CHANGE="2016-05-26T08:59:00+02:00">
  <SPEC-ATTRIBUTES>
    <ATTRIBUTE-DEFINITION-STRING IDENTIFIER="RC-2138659171" LONG-NAME="ReqIF.Name" LAST-CHANGE="2016-05-26T08:59:00+02:00">
      <TYPE>
        <DATATYPE-DEFINITION-STRING-REF>DT-ShortString</DATATYPE-DEFINITION-STRING-REF>
      </TYPE>
    </ATTRIBUTE-DEFINITION-STRING>
    <ATTRIBUTE-DEFINITION-XHTML IDENTIFIER="RC-2138842117" LONG-NAME="ReqIF.Text" LAST-CHANGE="2016-05-26T08:59:00+02:00">
      <TYPE>
        <DATATYPE-DEFINITION-XHTML-REF>DT-FormattedText</DATATYPE-DEFINITION-XHTML-REF>
      </TYPE>
    </ATTRIBUTE-DEFINITION-XHTML>
  </SPEC-ATTRIBUTES>
</SPEC-OBJECT-TYPE>
```

Redundant data is like LAST-CHANGE is deduplicated

13.2.2 StatementClasses

SpecIF	ReqIF
id	IDENTIFIER
title	LONG-NAME
description	DESC
propertyClasses[]	SPEC-ATTRIBUTES
changedAt	LAST-CHANGE

statementClass example:

in SpecIF:

```
"statementClasses":
[
  {
    "id": "SC-shows",
    "title": "SpecIF:shows",
    "description": [
      {
        "text": "Statement: Plan resp. diagram shows model element."
      }
    ],
    "revision": "1.1",
    "replaces": [],
    "instantiation": [
      "auto"
    ],
    "subjectClasses": [
      {
        "id": "RC-Diagram",
        "revision": "1.1"
      }
    ],
    "objectClasses": [
      {
        "id": "RC-Actor",
        "revision": "1.1"
      },
      {
        "id": "RC-State",
        "revision": "1.1"
      },
      {
        "id": "RC-Event",
        "revision": "1.1"
      }
    ],
    "changedAt": "2016-05-26T08:59:00+02:00"
  }
]
```

in ReqIF:

```
<SPEC-RELATION-TYPE IDENTIFIER="SC-shows" LONG-NAME="SpecIF:shows" DESC="Statement: Plan re
sp. diagram shows model element." LAST-CHANGE="2016-05-26T08:59:00+02:00">
</SPEC-RELATION-TYPE>
```

13.2.3 Resources

SpecIF	ReqIF
id	IDENTIFIER
title	LONG-NAME
class	SPEC-OBJECT-TYPE-REF
description	DESC
properties[]	VALUES
changedAt	LAST-CHANGE

resources example:

in SpecIF:

```
"resources":
[
  {
    "id": "Req-5ba3512b0000bca",
    "class":
    {
      "id": "RC-Requirement"
    },
    "properties": [
      {
        "class":
        {
          "id": "PC-Name"
        },
        "values":
        [
          "Minimum button size"
        ]
      },
      {
        "class":
        {
          "id": "PC-Description"
        },
        "values":
        [
          "\n
          <html:div xmlns:html=\"http://www.w3.org/1999/xhtml\">\n
          <html:p>\n
          <html:i>button size</html:i>\n
          MUST not be less than 20mm in diameter.\n
          </html:p>\n
          <html:object data=\"images/button-diameter.png\" type=\"image/png\">Diameter in different Forms</html:object>\n
          </html:p>\n
          </html:div>\n
          "
        ]
      }
    ],
    "changedAt": "2017-06-19T20:13:08+02:00"
  }
]
```

in ReqIF:

```
<SPEC-OBJECT IDENTIFIER="Req-5ba3512b0000bca" LONG-NAME="Minimum button size" DESC="" LAST-CHANGE="2017-06-19T20:13:08+02:00">
```



```
<TYPE>
  <SPEC-OBJECT-TYPE-REF>RC-Requirement</SPEC-OBJECT-TYPE-REF>
</TYPE>
<VALUES>
  <ATTRIBUTE-VALUE-STRING THE-VALUE="Minimum button size">
    <DEFINITION>
      <ATTRIBUTE-DEFINITION-STRING-REF>PC-Name</ATTRIBUTE-DEFINITION-STRING-REF>
    </DEFINITION>
  </ATTRIBUTE-VALUE-STRING>
  <ATTRIBUTE-VALUE-XHTML>
    <DEFINITION>
      <ATTRIBUTE-DEFINITION-XHTML-REF>PC-Description</ATTRIBUTE-DEFINITION-XHTML-
REF>
    </DEFINITION>
  <THE-VALUE>
    <xhtml:div>
      <xhtml:p>The
        <xhtml:i>button size</xhtml:i>
        MUST not be less than 20mm in diameter.
      </xhtml:p>
      <xhtml:p>
        <xhtml:object data="images/button-diameter.png" type="image/png">Di
iameter in different Forms</xhtml:object>
      </xhtml:p>
    </xhtml:div>
  </THE-VALUE>
</ATTRIBUTE-VALUE-XHTML>
</VALUES>
</SPEC-OBJECT>
```

13.2.4 Statements

SpecIF	ReqIF
id	IDENTIFIER
title	LONG-NAME
class	SPEC-RELATION-TYPE-REF
description	DESC
subject	SPEC-OBJECT-REF
object	SPEC-OBJECT-REF
changedAt	LAST-CHANGE

in SpecIF:

```
"statements":
[
  {
    "id": "RVis-Pln-5a4755dd0000bca801375293a62c90a8-ME1-5bd6bd890000bca8013739588a
3f43d6",
    "class":
    {
      "id": "SC-shows"
    },
    "changedAt": "2017-06-19T20:13:33+02:00",
    "subject":
    {
      "id": "Pln-5a4755dd0000bca801375293a62c90a8"
    },
    "object":
    {
```



```
    "id": "ME1-5bd6bd890000bca8013739588a3f43d6"  
  }  
}  
]
```

in ReqIF:

```
<SPEC-RELATION IDENTIFIER="RVis-Pln-5a4755dd0000bca801375293a62c90a8-ME1-5bd6bd890000bca801  
3739588a3f43d6" LONG-NAME="" DESC="" LAST-CHANGE="2017-06-19T20:13:33+02:00">  
  <TYPE>  
    <SPEC-RELATION-TYPE-REF>SC-shows</SPEC-RELATION-TYPE-REF>  
  </TYPE>  
  <VALUES>  
    <ATTRIBUTE-VALUE-STRING THE-VALUE="SpecIF:shows">  
      <DEFINITION>  
        <ATTRIBUTE-DEFINITION-STRING-REF>RC--669466474</ATTRIBUTE-DEFINITION-STRING  
-REF>  
      </DEFINITION>  
    </ATTRIBUTE-VALUE-STRING>  
  </VALUES>  
  <SOURCE>  
    <SPEC-OBJECT-REF>Pln-5a4755dd0000bca801375293a62c90a8</SPEC-OBJECT-REF>  
  </SOURCE>  
  <TARGET>  
    <SPEC-OBJECT-REF>ME1-5bd6bd890000bca8013739588a3f43d6</SPEC-OBJECT-REF>  
  </TARGET>  
</SPEC-RELATION>
```

13.3 Hierarchies

SpecIF	ReqIF
id	IDENTIFIER
resource	SPEC-OBJECT-REF
changedAt	LAST-CHANGE
nodes[]	CHILDREN

Examples:

in SpecIF:

```
"hierarchies": [  
  {  
    "id": "SH-Fld-5a5f54090000bca801375b04a668f1a7",  
    "resource": "Fld-5a5f54090000bca801375b04a668f1a7",  
    "changedAt": "2017-06-19T20:14:47+02:00",  
    "nodes": [  
      {  
        "id": "SH-Pln-27420ffc0000c3a8013ab527ca1b71f5",  
        "resource":  
          {  
            "id": "Pln-27420ffc0000c3a8013ab527ca1b71f5"  
          },  
        "changedAt": "2017-06-19T20:14:47+02:00"  
      },  
      {  
        "id": "SH-Pln-5a4755dd0000bca801375293a62c90a8",  
        "resource":  
          {  
            "id": "Pln-5a4755dd0000bca801375293a62c90a8"  
          }  
      }  
    ]  
  }  
]
```



```
        "id": "Pln-5a4755dd0000bca801375293a62c90a8"
      },
      "changedAt": "2017-06-19T20:14:47+02:00"
    },
    {
      "id": "SH-Pln-5a6cdea50000bca80137d6b2d6e8a3a0",
      "resource":
      {
        "id": "Pln-5a6cdea50000bca80137d6b2d6e8a3a0"
      },
      "changedAt": "2017-06-19T20:14:47+02:00"
    },
    {
      "id": "SH-Pln-5a7f99af0000bca8013754f2ef12d3e5",
      "resource":
      {
        "Pln-5a7f99af0000bca8013754f2ef12d3e5"
      },
      "changedAt": "2017-06-19T20:14:47+02:00"
    }
  ]
}
```

in ReqIF:

```
<SPEC-HIERARCHY IDENTIFIER="SH-Flid-5a5f54090000bca801375b04a668f1a7" LONG-NAME="" LAST-CHANGE="2017-06-19T20:14:47+02:00">
  <OBJECT>
    <SPEC-OBJECT-REF>Flid-5a5f54090000bca801375b04a668f1a7</SPEC-OBJECT-REF>
  </OBJECT>
  <CHILDREN>
    <SPEC-HIERARCHY IDENTIFIER="SH-Pln-27420ffc0000c3a8013ab527ca1b71f5" LONG-NAME="" LAST-CHANGE="2017-06-19T20:14:47+02:00">
      <OBJECT>
        <SPEC-OBJECT-REF>Pln-27420ffc0000c3a8013ab527ca1b71f5</SPEC-OBJECT-REF>
      </OBJECT>
    </SPEC-HIERARCHY>
    <SPEC-HIERARCHY IDENTIFIER="SH-Pln-5a4755dd0000bca801375293a62c90a8" LONG-NAME="" LAST-CHANGE="2017-06-19T20:14:47+02:00">
      <OBJECT>
        <SPEC-OBJECT-REF>Pln-5a4755dd0000bca801375293a62c90a8</SPEC-OBJECT-REF>
      </OBJECT>
    </SPEC-HIERARCHY>
    <SPEC-HIERARCHY IDENTIFIER="SH-Pln-5a6cdea50000bca80137d6b2d6e8a3a0" LONG-NAME="" LAST-CHANGE="2017-06-19T20:14:47+02:00">
      <OBJECT>
        <SPEC-OBJECT-REF>Pln-5a6cdea50000bca80137d6b2d6e8a3a0</SPEC-OBJECT-REF>
      </OBJECT>
    </SPEC-HIERARCHY>
    <SPEC-HIERARCHY IDENTIFIER="SH-Pln-5a7f99af0000bca8013754f2ef12d3e5" LONG-NAME="" LAST-CHANGE="2017-06-19T20:14:47+02:00">
      <OBJECT>
        <SPEC-OBJECT-REF>Pln-5a7f99af0000bca8013754f2ef12d3e5</SPEC-OBJECT-REF>
      </OBJECT>
    </SPEC-HIERARCHY>
  </CHILDREN>
</SPEC-HIERARCHY>
```


14 Non normative class definitions

The following domain definitions are not included in the current SpecIF release and are included as a perspective for future features and for informational purposes.

14.1 Domain 04: Automotive Requirements Engineering

14.1.1 Data types of domain 04: Automotive Requirements Engineering

title	id	revision	type	description
VDA:SupplierStatus	DT-SupplierStatus	1.1	xs:string	Enumerated values for VDA supplier-status VDA:notApplicable [V-SupplierStatus-0] VDA:toClarify [V-SupplierStatus-1] VDA:agreed [V-SupplierStatus-2] VDA:partlyAgreed [V-SupplierStatus-3] VDA:notAgreed [V-SupplierStatus-4]
VDA:OemStatus	DT-OemStatus	1.1	xs:string	Enumerated values for VDA oem-status VDA:notToEvaluate [V-OemStatus-0] VDA:toEvaluate [V-OemStatus-1] VDA:accepted [V-OemStatus-2] VDA:notAccepted [V-OemStatus-3]

14.1.2 Property classes of domain 04: Automotive Requirements Engineering

title	id	revision	dataType	description
VDA:SupplierStatus	PC-SupplierStatus	1.1	VDA:SupplierStatus	
VDA:SupplierComment	PC-SupplierComment	1.1	Plain or formatted Text	
VDA:OemStatus	PC-OemStatus	1.1	VDA:OemStatus	
VDA:OemComment	PC-OemComment	1.1	Plain or formatted Text	

14.1.3 Resource classes of domain 04: Automotive Requirements Engineering

title	id	revision	description
VDA:Diagram	RC-VDA_Diagram	1.1	A 'VDA:Diagram' is a Diagram with automotive-process-specific properties. Property classes: VDA:SupplierStatus [PC-SupplierStatus 1.1] VDA:SupplierComment [PC-SupplierComment 1.1] VDA:OemStatus [PC-OemStatus 1.1] VDA:OemComment [PC-OemComment 1.1]
VDA:Requirement	RC-VDA_Requirement	1.1	A VDA:Requirement is a Requirement with additional automotive-process-specific properties. Property classes: VDA:SupplierStatus [PC-SupplierStatus 1.1] VDA:SupplierComment [PC-SupplierComment 1.1] VDA:OemStatus [PC-OemStatus 1.1] VDA:OemComment [PC-OemComment 1.1]
VDA:Feature	RC-VDA_Feature	1.1	A 'VDA:Feature' is a Feature with automotive specific properties. Property classes: VDA:SupplierStatus [PC-SupplierStatus 1.1] VDA:SupplierComment [PC-SupplierComment 1.1] VDA:OemStatus [PC-OemStatus 1.1]

title	id	revision	description
			VDA:OemComment [PC-OemComment 1.1]

14.2 Domain 05: Agile Requirements Engineering

14.2.1 Resource classes of domain 05: Agile Requirements Engineering

title	id	revision	description
SCRUM:Epic	RC-Epic	1.1	An 'Epic' is a big user story and will be refined by user stories/requirements.

14.3 Domain 07: Issue Management

14.3.1 Data types of domain 07: Issue Management

title	id	revision	type	description
SpecIF:IssueStatus	DT-IssStatus	1.1	xs:string	Enumerated values for issue status open [V-IssStatus-0] assigned [V-IssStatus-1] in progress [V-IssStatus-2] closed [V-IssStatus-3] reopened [V-IssStatus-4] rejected [V-IssStatus-5]

14.3.2 Property classes of domain 07: Issue Management

title	id	revision	dataType	description
SpecIF:IssueStatus	PC-IssStatus	1.1	SpecIF:IssueStatus	

14.3.3 Resource classes of domain 07: Issue Management

title	id	revision	description
SpecIF:Issue	RC-Issue	1.1	An 'Issue' is a question to answer or decision to take which is worth tracking. Property classes: dcterms:identifier [PC-VisibleId 1.1] dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] SpecIF:IssueStatus [PC-IssStatus 1.1] SpecIF:Priority [PC-Priority 1.1] SpecIF:Responsible [PC-Responsible 1.1] SpecIF:DueDate [PC-DueDate 1.1]

14.4 Domain 08: BOM

14.4.1 Resource classes of domain 08: BOM

title	id	revision	description
SpecIF:BillOfMaterials	RC-BillOfMaterials	1.1	Root node of a bill of materials or product structure (sometimes bill of material, BOM or associated list) is a list of the raw materials, sub-assemblies, intermediate assemblies, sub-components, parts and the quantities of each needed to manufacture an end product. A BOM may be used for communication between manufacturing partners, or confined to a single manufacturing plant. Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1]

14.5 Domain 09: Variant Management

14.5.1 Data types of domain 09: Variant Management

title	id	revision	type	description
SpecIF:FeatureKind	DT-FeatureKind	1.1	xs:string	Enumerated values for feature kind Mandatory [V-FeatureKind-Mandatory] Alternative [V-FeatureKind-Alternative] Optional [V-FeatureKind-Optional] Or [V-FeatureKind-Or]
SpecIF:FeatureSelectionState	DT-FeatureSelectionState	1.1	xs:string	Enumerated values for feature selection state Unchecked [V-FeatureSelectionState-Unchecked] Checked [V-FeatureSelectionState-Checked] Forbidden [V-FeatureSelectionState-Forbidden] Recommended [V-FeatureSelectionState-Recommended]

14.5.2 Property classes of domain 09: Variant Management

title	id	revision	dataType	description
SpecIF:FeatureKind	PC-FeatureKind	1.1	SpecIF:FeatureKind	The kind of a feature used in a feature tree (mandatory, optional, alternative, or).
SpecIF:FeatureSelectionState	PC-FeatureSelectionState	1.1	SpecIF:FeatureSelectionState	The selection state for a feature used in a variant model.

14.5.3 Resource classes of domain 09: Variant Management

title	id	revision	description
SpecIF:FeatureModel	RC-FeatureModel	1.1	A 'Feature Model' is the root resource of a feature tree to structure the set of features for a system as a tree structure (hierarchy). Property classes: dcterms:identifier [PC-VisibleId 1.1] dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] SpecIF:LifeCycleStatus [PC-LifeCycleStatus 1.1]
SpecIF:VariantModel	RC-VariantModel	1.1	A 'Variant Model' is an instance of a feature model, where a subset of features is selected to define a concrete system variant. Property classes: dcterms:identifier [PC-VisibleId 1.1] dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] SpecIF:LifeCycleStatus [PC-LifeCycleStatus 1.1]
SpecIF:FeatureModelFeature	RC-FeatureModelFeature	1.1	A 'Feature Model Feature' is a feature used in a feature tree resp. feature model structure. Property classes: SpecIF:FeatureKind [PC-FeatureKind 1.1]

title	id	revision	description
			SpecIF:FeatureSelectionMode [PC-FeatureSelectionMode 1.1] SpecIF:Abbreviation [PC-Abbreviation 1.1]

14.6 Domain 10: Vocabulary Definition

14.6.1 Resource classes of domain 10: Vocabulary Definition

title	id	revision	description
SpecIF:TermResourceClass	RC-ResourceTerm	1.1	A term for resources (objects, entities) belonging to the SpecIF vocabulary Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] SpecIF:Origin [PC-Origin 1.1] SpecIF:LifeCycleStatus [PC-LifeCycleStatus 1.1]
SpecIF:TermStatementClass	RC-PredicateTerm	1.1	A term for statements (relations) belonging to the SpecIF vocabulary Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] SpecIF:Origin [PC-Origin 1.1] SpecIF:LifeCycleStatus [PC-LifeCycleStatus 1.1]
SpecIF:PropertyTerm	RC-TermProperty	1.1	A term for user-defined properties (attributes) belonging to the SpecIF vocabulary Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] SpecIF:Origin [PC-Origin 1.1] SpecIF:LifeCycleStatus [PC-LifeCycleStatus 1.1]
SpecIF:TermPropertyValue	RC-ValueTerm	1.1	A term for property values belonging to the SpecIF vocabulary Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] SpecIF:Origin [PC-Origin 1.1] SpecIF:LifeCycleStatus [PC-LifeCycleStatus 1.1]

14.6.2 Statement classes of domain 10: Vocabulary Definition

title	id	revision	description
SpecIF:isSynonymOf	SC-isSynonymousResource	1.0	Synonymous resource-term
SpecIF:isSynonymOf	SC-isSynonymousPredicate	1.0	Synonymous predicate-term
SpecIF:isSynonymOf	SC-isSynonymousProperty	1.0	Synonymous property-term
SpecIF:isSynonymOf	SC-isSynonymousValue	1.0	Synonymous value-term
SpecIF:isInverseOf	SC-isInverseOf	1.0	Two terms are the inverse of each other, such as 'contains' and 'isContainedBy'.

14.7 Domain 11: Testing

14.7.1 Data types of domain 11: Testing

title	id	revision	type	description
U2TP:Verdict	DT-TestVerdict	1.1	xs:string	A verdict is a predefined enumeration specifying the set of possible evaluations of a test case. None [V-Verdict-0] Pass [V-Verdict-1] Inconclusive [V-Verdict-2] Fail [V-Verdict-3] Error [V-Verdict-4]

14.7.2 Property classes of domain 11: Testing

title	id	revision	dataType	description
U2TP:Verdict	PC-TestVerdict	1.1	U2TP:Verdict	A verdict is a predefined enumeration specifying the set of possible evaluations of a test case.
ISTQB:ExpectedResult	PC-ExpectedResult	1.1		The predicted observable behavior of a component or system executing under specified conditions, based on its specification or another source. [After ISO 29119]
ISTQB:TestData	PC-TestData	1.1	Plain or formatted Text	Data created or selected to satisfy the execution preconditions and inputs to execute one or more test cases. [After ISO 29119]
U2TP:ReasonMessage	PC-TestResultReason	1.1		A textual note, describing a reason for a test result.
ISTQB:Precondition	PC-TestPrecondition	1.1	Plain or formatted Text	The required state of a test item and its environment prior to test case execution.
ISTQB:TestObject	PC-TestObject	1.1		The work product to be tested.

14.7.3 Resource classes of domain 11: Testing

title	id	revision	description
ISTQB:TestCase	RC-TestCase	1.1	A Test Case. Property classes: dcterms:identifier [PC-VisibleId 1.1] dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] ISTQB:Precondition [PC-TestPrecondition 1.1] U2TP:Verdict [PC-TestVerdict 1.1] SpecIF:LifeCycleStatus [PC-LifeCycleStatus 1.1] SpecIF:Priority [PC-Priority 1.1]
U2TP:TestStep	RC-TestStep	1.1	The smallest atomic (i.e., indivisible) part of a test case specification that is executed by a test execution system during test case execution. Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] ISTQB:TestData [PC-TestData 1.1] ISTQB:ExpectedResult [PC-ExpectedResult 1.1] U2TP:Verdict [PC-TestVerdict 1.1] U2TP:ReasonMessage [PC-TestResultReason 1.1]

title	id	revision	description
U2TP:TestModel	RC-TestModel	1.1	A set of test case definitions. Used as SpecIF hierarchy-root element. Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1]
ISTQB:TestSuite	RC-TestSuite	1.1	A set of test scripts or test procedures to be executed in a specific test run. Property classes: dcterms:title [PC-Name 1.1] dcterms:description [PC-Description 1.1] ISTQB:TestObject [PC-TestObject 1.1] U2TP:Verdict [PC-TestVerdict 1.1] U2TP:ReasonMessage [PC-TestResultReason 1.1]

14.7.4 Statement classes of domain 11: Testing

title	id	revision	description
ISTBQ:Verifies	SC-verifies	1.1	Statement: The test case (subject) verifies the requirement (object).

14.8 Domain 12: SpecIF Events

14.8.1 Data types of domain 12: SpecIF Events

title	id	revision	type	description
SpecIF:specifEventType	DT-SpecIfEventType	1.1	xs:string	The SpecIF Event Type. Resource created [V-SET-ResourceCreated] Resource updated [V-SET-ResourceUpdated] Resource deleted [V-SET-ResourceDeleted] Statement created [V-SET-StatementCreated] Statement updated [V-SET-StatementUpdated] Statement deleted [V-SET-StatementDeleted]

14.8.2 Property classes of domain 12: SpecIF Events

title	id	revision	data Type	description
SpecIF:apiURL	PC-APIURL	1.1	URL	The SpecIF API server URL.
SpecIF:project	PC-SpecIfProject	1.1	String[256]	The SpecIF project ID.
SpecIF:specifEventType	PC-SpecIfEventType	1.1	SpecIF:specifEventType	The SpecIF event type.
SpecIF:id	PC-SpecIfId	1.1	String[256]	The SpecIF element id.
SpecIF:revision	PC-SpecIfRevision	1.1	String[256]	The SpecIF element revision.
SpecIF:classId	PC-SpecIfClassId	1.1	String[256]	The SpecIF element class id.
SpecIF:classRevision	PC-SpecIfClassRevision	1.1	String[256]	The SpecIF element class revision.

Resource classes of domain 12: SpecIF Events

title	id	revision	description
SpecIF:specifEvent	RC-SpecIfEvent	1.1	<p>A SpecIF change event.</p> <p>Property classes:</p> <p>SpecIF:Origin [PC-Origin 1.1]</p> <p>SpecIF:project [PC-SpecIfProject 1.1]</p> <p>SpecIF:specifEventType [PC-SpecIfEventType 1.1]</p> <p>SpecIF:id [PC-SpecIfId 1.1]</p> <p>SpecIF:revision [PC-SpecIfRevision 1.1]</p> <p>SpecIF:classId [PC-SpecIfClassId 1.1]</p> <p>SpecIF:classRevision [PC-SpecIfClassRevision 1.1]</p>

15 References

15.1 SpecIF publications

- [KP15] Kaufmann, U., Pfenning, M. et al.: *10 Theses about MBSE and PLM*. PLM4MBSE Working Group Position Paper, GfSE, 2015. https://gfse.de/Dokumente_Mitglieder/ag_ergebnisse/PLM4MBSE/PLM4MBSE_Position_paper_V_1_1.pdf
- [Dungern14a] Dungern, O.v.: *Semantic Model-Integration for System Specification – Meaningful, Consistent and Viable*, 7. Grazer Symposium Virtuelles Fahrzeug, Graz, May 2014.
- [Dungern14b] Dungern, O.v.: *Übergreifende Konzeption von Geräten für die Gebäudeautomation – Methodik und Management*. TdSE - Tag des Systems Engineering der GfSE, Bremen, November 2014.
- [Dungern15] Dungern, O.v.: *Integration von Systemmodellen mit fünf fundamentalen Elementtypen*. TdSE - Tag des Systems Engineering der GfSE, Ulm, November 2015.
- [Dungern16a] Dungern, O.v.: *Von Anforderungslisten zu vernetzten Produktmodellen – am Beispiel der Gebäudeautomation*. REConf, March 2016 Unterschleißheim.
- [Dungern16b] Dungern, O.v.: *Semantic Model Integration for System Specification - Creating a Common Context for Different Model Types*. TdSE - Tag des Systems Engineering der GfSE, Herzogenaurach, Oktober 2016.
- [DU17] Dungern, O.v.; Uphoff, F.: *Der ‚Interaction Room‘ bahnt einen natürlichen Weg zu vernetzten Systemspezifikationen*. REConf, March 2017, München.
- [Dungern17] Dungern, O.v.: *How to Automatically Understand and Integrate System-models ... and how SpecIF can help*. GfSE EMEA Workshop Mannheim, September 2017.
- [MSDS17] Mochine, Ph.; Sünnetcioglu, A.; Dungern, O.v.; Stark, R.: *SysML-Modelle maschinell verstehen und verknüpfen*. TdSE - Tag des Systems Engineering der GfSE, Paderborn, November 2017.
- [Alt18] Alt, O.: *SpecIF - Die kommende vielschichtige Datenquelle für Spezifikationsdaten*. Fachgruppentreffen GI-RE, Nürnberg, November 2018.
- [Dungern19] Dungern, O.v.: *Model-Integration with SpecIF*. ProSTEP ivip e.V. SysML-Workflow-Forum November 2019.
- [DA20] Dungern, O.v.; Alt, O.: *Specification Integration Facility - Wozu braucht man SpecIF neben SysML?*. TdSE - Tag des Systems Engineering der GfSE, online, November 2020.
- [Dungern21] Dungern, O.v.: *Integrate BPMN and Archimate Models using SpecIF*. TdSE - Tag des Systems Engineering der GfSE, online, November 2021.
- [SpecIF21] GfSE: *SpecIF - Homepage*. <https://specif.de/en/> with <https://specif.de/en/#literature>

15.2 Standards

- [ReqIF16] OMG: *Requirements Interchange Format (ReqIF) - Version 1.2*. OMG Document Number: formal/2016-07-01, 2016. <https://www.omg.org/spec/ReqIF/1.2/PDF>
- [UML11] OMG: *OMG Unified Modeling Language™ (OMG UML), Superstructure - Version 2.4.1*. OMG Document Number: formal/2011-08-06, 2011. <https://www.omg.org/spec/UML/2.4.1/Superstructure/PDF>
- [SysML19] OMG: *OMG Systems Modeling Language (OMG SysML™) - Version 1.6*. OMG Document Number: formal/19-11-01, 2019. <https://www.omg.org/spec/SysML/1.6/PDF>
- [BPMN11] OMG: *Business Process Model and Notation (BPMN) - Version 2.0*. OMG Document Number: formal/2011-01-03, 2011. <https://www.omg.org/spec/BPMN/2.0/PDF>
- [DI06] OMG: *Diagram Interchange - Version 1.0*. OMG Document Number: formal/06-04-04, 2006. <https://www.omg.org/spec/UMLDI/1.0/PDF>
- [DD15] OMG: *Diagram Definition (DD) - Version 1.1*. OMG Document Number: formal/2015-06-01, 2015. <https://www.omg.org/spec/DD/1.1/PDF>
- [SVG11] W3C: *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. W3C Recommendation 16 August 2011. <https://www.w3.org/TR/SVG11/>
- [JS21] JSON Schema: *JSON Schema Specification 2019-09*. 2019-09. <https://json-schema.org/specification.html>
- [XML08] W3C: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation 26 November 2008. <https://www.w3.org/TR/REC-xml/>
- [Archimate19] The Open Group: *ArchiMate® 3.1 Specification, a Standard of The Open Group*. <https://pubs.opengroup.org/architecture/archimate3-doc/toc.html>
- [FMC21] Wendt, S.: *FMC - Home of Fundamental Modeling Concepts*. 2021. <http://fmc-modeling.org/>
- [DC20] DublinCore: *DCMI Metadata Terms*. 2020-01-20. <https://www.dublincore.org/specifications/dublin-core/dcmi-terms/>
- [IREB21] IREB: *CPRE Glossary 2.0*. 2021. <https://www.ireb.org/en/cpre/cpre-glossary/>
- [OSLC21] OSLC: *OSLC Specifications*. 2021. <https://open-services.net/specifications/>
- [XHTML00] W3C: *XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition)*. W3C Recommendation 26 January 2000, revised 1 August 2002 superseded 27 March 2018. <https://www.w3.org/TR/xhtml1/>
- [OpenAPI21] Open API Initiative: *OpenAPI Specification v3.1.0*. Published 15 February 2021. <https://spec.openapis.org/oas/v3.1.0>



- [MOF19] OMG: *OMG Meta Object Facility (MOF) Core Specification - Version 2.5.1.*
OMG Document Number: formal/2019-10-01, 2019.
<https://www.omg.org/spec/MOF/2.5.1/PDF>
- [MDA21] OMG: *Model Driven Architecture (MDA).*
<https://www.omg.org/mda/index.htm>