Requirements Interchange Format (RIF)

RECOMMENDATION

Requirements Interchange Format (RIF), PSI 6
Version 1.2

ProSTEP iViP

Abstract

Requirements management has been established in the industry as a primary method to get flawlessly from the early product target definitions over the implementation up to the validation and testing phase. Despite its cross-functional nature requirements management is a key factor for successful partner integration within the development cycle. Manufacturers specify requirements for their entire product as well as the aggregation of systems/components and their suppliers provide system specifications describing how their product will meet these needs. In order to manage the complexity growths due to the increase of requirements and requirement specifications, especially with regards to mechatronics, manufacturers and suppliers introduced requirements management systems. These systems are used within an organizational context but can not really bridge the gap between manufacturers and suppliers, as only tool specific exchange formats and methods exist.

In order to address the circumstance of the unavailability of a generic requirements exchange format the "Simulation and Tools" Group of the HIS (Hersteller Initiative Software) specified the Requirement Interchange Format (RIF). This specification was then handed over to the ProSTEP iViP Association to be published as ProSTEP iViP Recommendation. This Recommendation will serve as a basis for a future international standardization of the RIF to enable its broad application.

This ProSTEP iViP Recommendation covers the use cases to be supported, the format specification as data model description and additional implementation details to be considered.

Disclaimer

ProSTEP iViP Recommendations (PSI Recommendations) are recommendations that are available for general use. Anyone using these recommendations is responsible for ensuring that they are used correctly.

This PSI Recommendation gives due consideration to the prevailing state-of-the-art at the time of publication. Anyone using PSI Recommendations must assume responsibility for his or her actions and acts at their own risk. The ProSTEP iViP Association and the parties involved in drawing up the PSI Recommendation assume no liability whatsoever.

We request that anyone encountering an error or the possibility of an incorrect interpretation when using the PSI Recommendation should contact the ProSTEP iViP Association (psi-issues@prostep.com) immediately so that any errors can be rectified.

Copyright

Table of Contents

Figures

1 Overview

1.1 Introduction

Requirements Management has been an integral part of the development process in various industries (especially in the military, aeronautical or the medical device industry) for years. Other industries have been adopting these techniques recently.

In the automotive industry, where the initiative for this interchange format started, the term “Requirements Management” (RM) found its entry around 1999. Up to then, car manufacturers and the supplier industry already took care of their requirements but did not make use of the advantages of modern Requirements Management. As Requirements Management spread in the automotive industry, more and more car manufacturers (OEM's) and suppliers have been applying Requirements Management and making use of Requirements Management tools. Large improvements have been made in these organizations. Now with a modern RM in place on both sides (OEMs and suppliers), it became obvious that RM should not stop at company borders but should bridge this gap.

For technical and organizational reasons, OEM and supplier will not be able to work on the same RM database. Because of that, a smart exchange mechanism needed to be found to exchange RM data between OEM and supplier without loosing the advantage of modern Requirements Management at the organization borders. RIF was created to support such an exchange of information to bridge the gap.

1.2 Who should read this document

This document is created to inform

- persons interested in exchanging RM data between organizations that do not have a possibility to share the same RM database.
- RM tool vendors who want support RIF with export and import interfaces for their RM tools.

1.3 Conventions

In this document, the following specific semantics are used for requirements (taken from Request for Change RFC 2119 from the Internet Engineering Task Force IETF).

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. Note that the requirement level of the document in which they are used modifies the force of these words.

- MUST: This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
- MUST NOT: This phrase, or the phrase „SHALL NOT“, means that the definition is an absolute prohibition of the specification.
- SHOULD: This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED" mean that there may ex-ist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed be-fore implementing any behavior described with this label.
- MAY: This word, or the adjective „OPTIONAL”, means that an item is truly optional. One ven-dor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation, which does not include a particular option, MUST be prepared to interoperate with another implementation, which does include the option, though perhaps with reduced functionality. In the same vein an implementation, which does include a particular option, MUST be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides.)

1.4 Structure of this Document / How to read this Document

Chapter 1 of this document gives an overview why RIF is needed, who worked on it and how it was created.
Chapter 2 describes the use cases that are to be supported by RIF. These use cases led to require-ments on the specification of RIF.
Chapter 3 introduces RIF by giving a description of the RIF information model.
Chapter 4 contains implementation details for RIF which are mostly related to XML. This chapter is thought for all people who have to implement interfaces for writing and reading RIF files.
Appendix A gives references on further RIF related resources such as the RIF XML Schema.
Appendix B contains a glossary explaining keywords and acronyms.
Appendix C contains a table on the proposed mapping of RIF elements to tool elements of certain well-known RM tools.
Appendix D finally contains a table with the history of changes of the RIF model.

1.5 Why we need an Interchange Format for Requirements

Gathering and documenting the requirements for the development of a system is an integral part in each system development process. The traditional approach to this task is quite simple: just write down the requirements into a document and call it "specification". Experience shows that doing the things this way leads to trouble in the later phases of the project. Incomplete requirements often cause changes in the deliverables. These changes are hard to do and expensive because they often do not fit to the initial concepts. Again, these changes are poorly documented.

Requirements Engineering (RE) and Requirements Management offer methods and tools to overcome this problem. In the beginning, they were used to develop large and safety critical systems like aircrafts or power plants. The increasing complexity of other products (e.g. software systems, cars) makes these methods more and more attractive for the development process of these products. Tools developed to support the RE/RM process focused on the management of all requirements for a project in a special database. All partners working on the requirements usually need direct access to this da-tabase. For documentation purposes, it was sufficient to export the requirements into a document that can be easily archived.

Increased complexity of the products and specialization of the partners require closer collaboration of the partners and lead to more and more specialized knowledge on the development tasks for all partners. If one partner also collaborates with competitors of the other partner, a common database is no longer sufficient for the interests of the partners. An obvious solution to this problem is to have the partners run their own RE processes and manage the different levels of requirements in their own databases. Doing things this way makes it necessary to interchange the requirements between the partners. Considering the features of current RM tools, this is usually done by the exchange of documents.

The traditional interchange process for requirements in the automotive industry starts with the user requirements that are documented by the OEM and delivered to the supplier. In most cases, the exchange of the requirements across company borders is only possible using one of the following mechanisms:

– print out the relevant information from the RM tool and send it to the partners (worst scenario)
– export the relevant information from the RM tool into a text processor readable format (most likely Microsoft® Word)
– if both sides use the same RM tool: hope that the RM tool vendor supports an offline exchange of RM data and use this one.

None of the above is satisfying because valuable tool-internal structural information such as traceable links is lost.

The supplier reads the user requirements, evaluates them and produces the system requirements as an answer to the user requirements of the OEM. If user and system requirements are exchanged as documents, working with these documents is often very time consuming, as the document structures may be completely independent. Comparing the system requirements with the user requirements is done by manually marking corresponding information. Due to the fact that the result of this work is mostly written into the documents, it is very hard to reuse them and identify the differences of new ver-sions of the documents that might appear during this process. It is evident to say that this way of working is not satisfying for all partners and not adequate to the opportunities that modern tools could offer.

1.6 Objectives of RIF

The fact that more and more OEMs and suppliers use RM tools to manage their requirements leads to a situation where a smart handling of requirements across company and tool borders should be possible. The most important thing that is missing is an exchange format which is independent from a special RM tool and supported by all vendors of RM tools. The Requirements Interchange Format (RIF) defines such a tool-independent exchange format. With the help of RM tools supporting RIF, it will be possible to bridge the gap between companies that cannot share a common RM database:

– The collaboration between partners is improved resulting from the benefits of applying RM methods across company borders.
– The partners do not have to use the same RM tool.

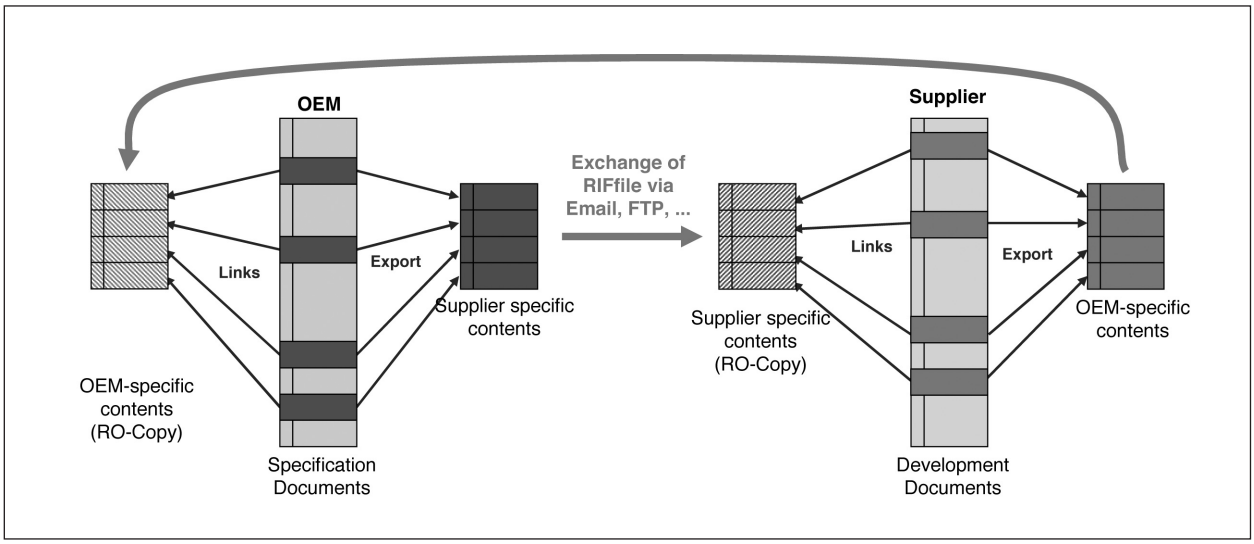


Figure 1: Exchange of requirements between two RM databases using RIF

Figure 1 shows a sample scenario for the exchange of requirements between two RM tools using RIF for the transport of information. Other scenarios can also be supported by RIF. The situation shown in the figure assumes that the OEM identifies and marks the requirements relevant for the supplier in his RM database. These requirements are exported to a RIF file that is sent to the supplier using traditional file transfer tools (e.g. email, ftp). Identifiers for the exported requirements are stored in the OEM RM database to support later re-import and -export. The supplier imports the RIF file into a separate repository in his RM database, starts to identify the corresponding system requirements in his database and adds new system requirements where necessary. These system requirements are then exported using RIF. The RIF file is sent back to the OEM and imported into his RM database. Since links to other requirements are maintained in the RIF file by including the identifiers of linked requirements, the links between corresponding user and system requirements in the OEM's RM database can be generated automatically.

The complete process can be repeated several times using an update mode for the repository that keeps the imported information. Dependant on the features of the RM tools, changes in the imported information can be easily made visible.

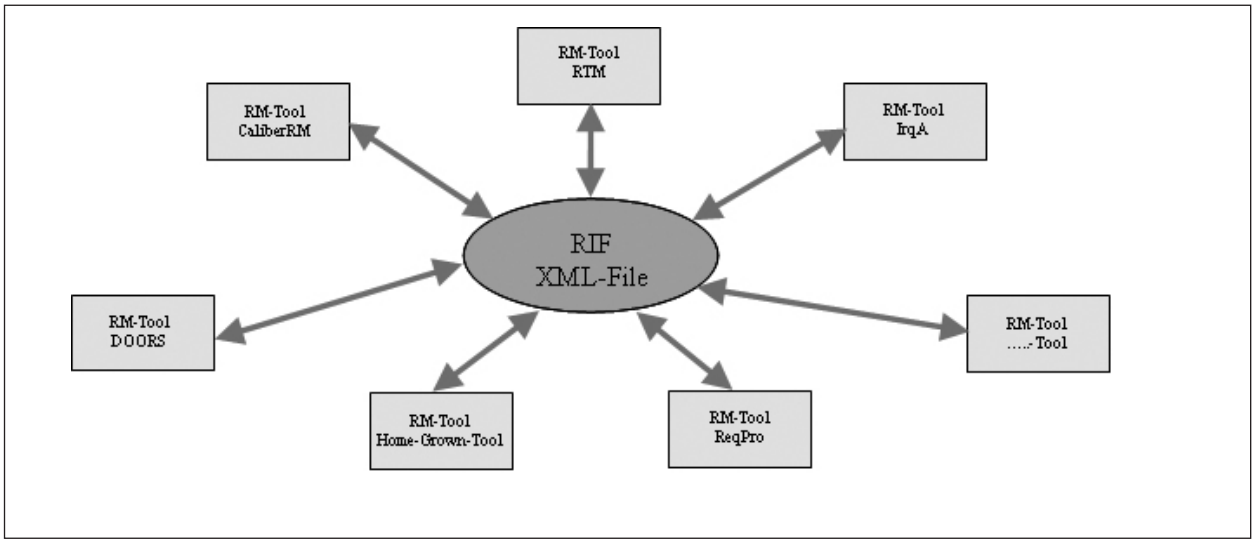


Figure 2: Use of RIF for the interchange of requirements between different tools

Figure 2 illustrates the usage of RIF for the interchange of information between different tools. One of the basic ideas of RIF is to offer the opportunity to store the information that can commonly be handled by all the tools. In order to avoid inconsistent usage of RIF model elements by different tools (the same RIF element could be used for different types of data structures), an additional mapping table is introduced. It defines how the different tools store their information in RIF data objects. Using this design, the file format description can be kept very stable concerning tool changes whereas the mapping table may have to be adjusted to enhanced tool capabilities. In principal, the definition of an interchange format cannot solve the problem of different tools supporting different types of information. The approach of HIS makes it possible that the exchange of information between different installations of the same tool can be done with a standardized format and that the same format can be used for exchange of information between different tools. In this case, the mapping table becomes important. The mapping describes how information is handled between the different tools. This information can be important to assess the impact of unsupported data structures and types in each tool when interchanging requirements in a project.

1.7 RIF Partners

The initial work on RIF was done by the members of the HIS group and additional partners that were associated for this project. The HIS group is the association of the vehicle manufacturers Audi AG, BMW Group, Daimler AG, Porsche AG and Volkswagen AG to bundle their activities for standard software modules, process maturity levels, software test, software tools and programming of control units. The common goal is to achieve and use joint standards. The group that is working on the initial release of RIF consists of the HIS members and Adam Opel AG, Conti Temic microelectronic GmbH, Carmeq GmbH and the HOOD Group.

2 Use Cases Supported by RIF

2.1 Overview: Main Use Cases

Four main use cases have been identified which are listed below. Use case UC-2 is a high summary use case and described in more detail through UC-3, UC-4 and UC-5:

UC-1 External creation of a complete specification

UC-2 Customer and supplier exchanging specifications
- UC-3 Send first version of a specification to a partner
- UC-4 An answer to (parts of) a previously received specification is sent
- UC-5 Send an updated version of a specification to a partner

UC-6 External review of a specification

UC-7 Proof reading of a specification

UC-8 Documentation generation

The first use case (UC-1) describes the situation where a specification is completely written by an external partner. In this case, all information can be modified after the document was exchanged.

UC-2 and its sub use cases focuses on the coordination of the specified content between two partners. In this process, the user requirements usually are specified by the OEM (partner 1) and sent to the supplier (partner 2). The supplier assesses the requirements in order to either agree to the requirements or give detailed feedback on individual topics. This results in a commented version of the specification that is sent back to the OEM who again assesses the feedback and adds his comments in the document. This process can be done several times until a common opinion on the requirements is reached. In this process, each partner has its own information objects where requirements and the feedback are stored. It is important that each partner is allowed to modify only the information objects he is responsible for.

UC-6 and UC-7 support quality ensuring activities related to exchanged specifications.

RIF can also be used to generate documentation and XML technologies like XSLT can be used for this purpose. This is covered by UC-8.

The template used for specifying the succeeding uses cases document is by Alistair Cockburn.

2.2 The Use Cases in Detail

2.2.1 Use Case: UC-1 External Creation of a complete Specification

CHARACTERISTIC INFORMATION

Goal in Context:
A company (customer) wants one or more other companies (specification writer) to write a specification for a system or subsystem that the customer wants to develop or wants to have developed by a sub-contractor or supplier. The delivered specification parts (from the various specification writers) are loaded into the RM-Tool Database of the customer.

Scope:	The exchange of the requirements data
Level:	Summary.
Preconditions:	The two parties have a contract in place. The specification writer knows which SuD or which parts of the SuD should be described. The customer has a RM-Tool installed.
Success End Condition:	The completed Requirements Specification is accessible in the customers RM-Tool-database.
Failed End Condition:	No Specification in the Customers RM-Tool-Database.
Primary Actor:	The "Customer" who wants to have the Specification written.
Trigger:	The Customer generates a Structure for the Specification and sends it to the Specification Writer.

MAIN SUCCESS SCENARIO

Step 1: Generate Specification Structure (in RM-Tool of Customer)
Step 2: Export (parts of) Specification Structure from RM-Tool into Exchange Files.
Step 3: Transfer this files to the Specification Writers
Step 4: Specification Writer creates (drafts of) the Specification in Exchange File Format.
Step 5: Specification Writer sends Exchange File (complete or draft Specification) to Customer.
Step 6: Customer loads the complete or draft Specification into his RM-Tool-Database.
Step 7: The previous version of the Specification in the customers RM-Tool-Database has to be merged with the latest version of the specification; a change history (of the changes between these two versions) has to become visible and shall be kept. It is visible, which specification writer produced which part of which Specification.
Step 8: The complete Specification is in Customers RM-Tool-Database.

EXTENSIONS

Step 4.1: In the main success scenario, nothing is said whether the Specification Writer uses a RM-Tool or not. It shall be possible for the Specification Writer to import the Exchange File into a RM-Tool, develop the Specification and export it into an Exchange File that can be sent to the Customer.
Step 9.1: If the Specification Writer uses a RM-Tool and gets a commented or corrected version of the Specification from the Customer, he then shall be able to merge this commented and corrected version with the information in his RM-Tool-Database.
Step 2.1: Specification Structure is sent in parts to various Specification Writers.
Step 5.1: Various Specification Writers are sending back their parts of the Specification.
Step 7.1: The various parts of the Specification are put together to re-build one, single Specification, while each Specification Writer and his contributed parts to the Specification must be identifiable.

SUB-VARIATIONS

empty

RELATED INFORMATION (optional)

empty

2.2.2 Use Case: UC-2 Customer and Supplier exchanging Specifications

CHARACTERISTIC INFORMATION

Goal in Context:
A company (customer) wants to exchange (parts of) Customer Requirements Specifications (CRS; Lastenheft) with one or more other companies (supplier). ("Lasten-/Pflichtenheftaustauschprozess").

After a series of exchanges of requirements data in specifications (CRS, SysRS) from the customer to the suppliers and backwards, the customer has in his RM-Tool-Database his CRS (with comments from the suppliers), the various SysRSs from the various suppliers (with the comments from him, the customer) and the links between the requirements data in the CRS to the requirements data in the various SysRS.

Scope:	The exchange of the requirements data.
Level:	High Summary.
Preconditions:	The two parties have a contract in place. The two parties have a RM-Tool installed (or at least are able to emulate the functionality of state-of-the-art RM-Tools).
Success End Condition:	The CRS with comments and the SysRSs of the various suppliers with relations between CRS and SysRSs are in the customers RM-Tool-database.
Failed End Condition:	No SysRS is in the customers RM-Tool-Database.
Primary Actor:	The Customer.
Trigger:	The Customer exports his CRS and sends it to the supplier.

MAIN SUCCESS SCENARIO

Step 1: Customer sends (parts of) his CRS from his RM-Tool to (one or more) Supplier.
This might be a new CRS or an update (changes, additions, deletions, structural changes) of a previously sent CRS.

Step 2: Supplier comments (accepts, refuses, counter-proposes, comments, questions) the CRS; Supplier creates/changes SysRS and creates relations between entries in CRS and entries in SysRS).
This might be a new SysRS or an update (changes, additions, deletions, structural changes) of a previous sent version

Step 3: Supplier sends (parts of) CRS with his comments, his SysRS and the Relations to the CRS to the Customer.

Step 4: Customer imports data from Suppliers into RM-Tool.

Step 5: The CRS with comments and the SysRSs of the various suppliers with Relations between CRS and SysRSs are in the customers RM-Tool-database.

More than one customer and more than one supplier is possible in all steps.

EXTENSIONS

Ext 1: Extension to Step 5: Customer wants to comment the SysRS

Step 5.1: Customer adds comments against SysRS (accepts, refuses, counter-proposes, comments, questions) and sends it back to the Supplier(s)

Step 5.2: Supplier does step 2 from main scenario

SUB-VARIATIONS

empty

RELATED INFORMATION (optional)

Subordinate Use Cases: UC-3; UC-4; UC-5

2.2.3 Use Case: UC-3 Send first version of a specification to a partner

CHARACTERISTIC INFORMATION

Goal in Context:

Partner1 wants to send a first version of a specification or parts of a specification to one or more partner.
The specification (or parts of it) is in the RM-Tool-database of the partner.

Scope:	The exchange of the Requirements Data.
Level:	Summary.
Preconditions:	The two parties have a contract in place. The two parties have a RM-Tool installed (or at least are able to emulate the functionality of state-of-the-art RM-Tools).
Success End Condition:	The Specification (or parts of it) is in the RM-Tool-Database of the Partner.
Failed End Condition:	No Specification is in the Partners' RM-Tool-Database.
Primary Actor:	Partner1.
Trigger:	The Customer exports his CRS and sends it to the Supplier.

MAIN SUCCESS SCENARIO

Step 1: Partner1 exports a Specification or selected part of the Specification from RM-Tool into an Exchange File.
Step 2: Transfer of Exchange File from Partner1 to Partner2 (and probably further more partners).
Step 3: Partner2 (and further partners) imports the Specification into his RM-Tool.
Step 4: The Specification (or parts of it) is in the RM-Tool-Database of Partner2 (and further partners).

EXTENSIONS

empty

SUB-VARIATIONS

empty

RELATED INFORMATION (optional)

Superordinate Use Case: UC-2

2.2.4 Use Case: UC-4 An Answer is sent to (parts of) a previously received Specification

CHARACTERISTIC INFORMATION

Goal in Context:
A company (Partner 2) received a Specification from Partner1. Partner2 loaded this Original Specifica-tion into his RM-Tool and now answers to this Specification with comments and/or an Answer Specifi-cation with Relations between the Original Specification and the Answer Specification.

The comments (with information from which partner), the Answer Specification and the Relations be-tween the Original Specification and the Answer Specification are visible in Partner1's RM-Tool-database.

Scope:	The exchange of the Requirements Data.
Level:	Summary.
Preconditions:	The two parties have a contract in place. The two parties have a RM-Tool installed (or at least are able to emulate the functionality of state-of-the-art RM-Tools). Partner2 received a Specification from Partner1. Partner2 has this Specification in his RM-Tool.
Success End Condition:	The comments (with information from which partner), the Answer Specification and the Relations between the Original Specification and the Answer Specification are visible in Partner1's RM-Tool-database.
Failed End Condition:	No answer is in the Partner1's RM-Tool-Database.

Primary Actor: Partner2.
Trigger: Partner2 comments on Specification from Partner1.

MAIN SUCCESS SCENARIO

Step 1: Partner2 adds comments (accepts, refuses, counter-proposes, comments, questions) against Original Specification and/or Partner2 creates an Answer Specification and establishes Relations between Objects of the Original Specification and Objects of the Answer Specification.
Step 2: Partner2 exports Original Specification with comments, Answer Specification and Link information from RM-Tool into an exchange file.
Step 3: Transfer of Exchange File from Partner2 to Partner1.
Step 4: Partner1 imports Original Specification with comments, Answer Specification and Link information from Partner2 (and perhaps further partners) into his RM-Tool.
Step 5: The comments, the Answer Specification and the Links between the Original Specification and the Answer Specification are visible in Partner1's RM-Tool-database. Partner1 can distinguish in his database between the answers from the various partners.

EXTENSIONS
empty

SUB-VARIATIONS
empty

RELATED INFORMATION (optional)
Superordinate Use Case: UC-2

2.2.5 Use Case: UC-5 Send an updated version of a specification to a partner

CHARACTERISTIC INFORMATION

Goals in Context:
Partner1 wants to send an updated versions of a previous sent Specifications (CRS or SysRS) or parts of the Specification to one or more partners (the various partner should receive the same part of the Specification as they received previously). The partner might have commented the previous sent version of the Specification and made Links to other parts of his RM-Tool-Database.

The updates of the Specification shall become visible in the partners RM-Tool without destroying previously made comments and Links.

Scope: The exchange of the requirements data.
Level: Summary.
Preconditions: The two parties have a contract in place.
The two parties have a RM-Tool installed (or at least are able to emulate the functionality of state-of-the-art RM-Tools).
The two parties have already exchanged a previous version of the specification.
Success End Condition: The Updated Specification and the changes to the previous version of the Specification are visible in the partners RM-Tool-database. Links from Requirements of the Previous Specification to other parts of the RM-Tool-Database of the partner are kept for the (identical) Requirements of the Updated Specification.
Failed End Condition: No Updated Specification is in the customers RM-Tool-Database; Relations are not transferred/copied from the previous to new version of the specification.

Primary Actor:	Partner1.
Trigger:	Partner1 exports an updated Specification and sends it to the other partner.

MAIN SUCCESS SCENARIO

Step 1: Partner1 updates Specification (changes, additions, deletions, structural changes).
Step 2: Partner1 exports Updated Specification into Exchange File;
Step 3: Transfer of Exchange File from Partner1 to Partner2;
Step 4: Import of changed Specification at Partner2 side into his RM-Tool.
Step 5: Changed Specification is visible in Partner2's RM-Tool without overwriting/deleting comments and links.
Step 6: Changes between original version and new version of the Specification are visible at Partner2 side RM-Tool.

EXTENSIONS

empty

SUB-VARIATIONS

empty

RELATED INFORMATION (optional)

Superordinate Use Case: UC-2

2.2.6 Use Case: UC-6 External review of a specification

CHARACTERISTIC INFORMATION

Goal in Context:

Customer sends Specification to one or more Reviewers. Reviewers send commented Specification back to Customer. The comments of the various Reviewers have to be visible in the RM-Tool, so that the Specification can be consolidated accordingly to the Review comments.

Scope:	The exchange of the Requirements Data.
Level:	Summary.
Preconditions:	Customer and Reviewer have a RM-Tool installed (or at least are able to emulate the functionality of state-of-the-art RM-Tools).
Success End Condition:	The comments of the various reviewers are visible in the customers RM-Tool database close to the commented area of the Specification.
Failed End Condition:	t.b.d.
Primary Actor:	Customer.
Trigger:	Customer sends out Specification for Review.

MAIN SUCCESS SCENARIO

Step 1: Customer sends Specification to one or more Reviewers.
Step 2: Reviewers comment Specification (set status, comment, question).
Step 3: Reviewers send back commented Specification.
Step 4: Import of the comments of the various Reviewers into Customers RM-Tool.
Step 5: Comments of the various Reviewers are visible in the customers RM-Tool database close to the commented area of the Specification, so that the Specification can be consolidated according to comments.

EXTENSIONS

empty

SUB-VARIATIONS
empty

RELATED INFORMATION (optional)
empty

2.2.7 Use Case: UC-7 Proof reading of a specification (Corrector Use Case)

CHARACTERISTIC INFORMATION

Goal in Context:
Customer wants proof reader to correct specification.

Scope:	The exchange of the Requirements Data.
Level:	Summary.
Preconditions:	Customer and Proof Reader have a RM-Tool installed (or at least are able to emulate the functionality of state-of-the-art RM-Tools).
Success End Condition:	The corrected versions are available in the customers RM-Tool so that reconciliation is possible.
Failed End Condition:	t.b.d.
Primary Actor:	Customer.
Trigger:	Customer sends Specification to Lectors.

MAIN SUCCESS SCENARIO
Step 1: Customer sends Specification to Proof Readers.
Step 2: Proof Readers make changes to the Specification.
Step 3: Proof Readers send corrected versions of Specification back to customer.
Step 4: Import of corrected Specifications into RM-Tool.
Step 5: The various corrected Specifications and the Original Specification can be viewed close to each other.
Step 6: Customer can reconcile Specifications.

EXTENSIONS
empty

SUB-VARIATIONS
empty

RELATED INFORMATION (optional)
empty

2.2.8 Use Case: UC-8 Documentation generation

CHARACTERISTIC INFORMATION

Goal in Context:
Customer wants an editable or printable document that contains requirements from a RIF file.

Scope:	The exchange of the Requirements Data.
Level:	Summary.
Preconditions:	Customer has a RIF file from which he wants to generate documentation from.
Success End Condition:	A document in a different format is generated.
Failed End Condition:	
Primary Actor:	Customer.
Trigger:	Customer needs an editable or printable version of the requirements.

MAIN SUCCESS SCENARIO

Step 1: Customer creates a document from requirements contained in a RIF file.

EXTENSIONS

empty

SUB-VARIATIONS

empty

RELATED INFORMATION (optional)

empty

3 RIF – Format and Interfaces

This chapter describes

- information on the methodology used for modeling the RIF exchange file format that is necessary to understand the model,
- the RIF-model itself,
- the workflow of importing and exporting RIF files that is necessary to implement RIF interfaces for RM tools.

The RIF-model is the basis for generating an XML-Schema that defines valid RIF exchange files. The conversion of the model into an XML-Schema and other XML related issues are addressed in Chapter 4.

3.1 RIF Modeling

3.1.1 Introduction: Why modeling RIF?

The data structures needed for storing and transferring requirements together with additional information (e.g. data types, links, access rights etc.) are complex due to many different relationships among the various data within the exchange file. A format that wants to enable its users to transport such complex data and relations needs adequate structures to depict this information. A means of handling this complexity is the adoption of graphically modeling the information elements and their relationships. The graphical RIF-model is based on UML class diagrams and thus straight-forward to understand. Using appropriate tools, it is possible to automatically convert the graphical RIF-model into an XML-Schema that can subsequently be used by RM tool vendors to validate the correctness of an RIF exchange file.

3.1.2 Information Elements and Information Types

An exchange file (EF) that is in accordance with the RIF Exchange File Format (EFF) contains a large amount of different data and information. From the point of view of an RM tool, the atomic unit of this data and information is the information element (e.g. a requirements text, an attribute value, a relation that links two requirements).

Information elements have certain common characteristics and can thus be categorized or typed. An *information type* is a category of information elements with the same properties in terms of e.g. element attributes or relationship to other information elements. An *information element* can thus be seen as an instantiation of an *information type*. Please note that there can be multiple instantiations of *information elements* from the same *information type* that differ from each other in the concrete values of element attributes and/or in the concrete relationships to other information elements.

The RIF-model is located on the same abstraction level as *information types*. It specifies and describes the different kind of *information types* as well as their relationships to each other.

As known from UML classes, an *information type* (IT) has the following characteristics:

- It has a name (the *IT name).*
- It is either *abstract or concrete.*
- It can have an arbitrary number of *(IT) attributes.*
- It can have additional *tags.*

The *IT name* is the identity of the information type and is unique within the RIF-model.

The property of being abstract means that information elements of this information type never appear in the RIF exchange file. *Abstract* information types are used internally for better structuring of the model. Information elements that can appear in the RIF exchange file are instantiated from *concrete* (non-abstract) information types.

Please note that in the graphical RIF-model, *abstract* information types are indicated by writing their names with *italic* letters.

IT attributes (not to be confused with requirement attributes!) contain data of a specified type (e.g. integer, real, string) that is associated with the information type.

Tags specify special properties of an information type with regard to the XML-Schema generation process. They are only relevant for certain XML related issues. These are described in chapter 4.

3.1.2.1 Relationships between Information Types

In the RIF-model, relationships can occur between exactly two *information types*. There are no n-ary relationships. A relationship between two *information types* means that two *information elements* that are instantiated from these two *information types* can have the specified relationship.
These relationships are one of the following:

- Aggregation
- Generalization
- Association

In the following, the explanation of the possible relationships is additionally given on the level of concrete *information elements* in order to simplify the understandability.

Please assume that IT1 and IT2 are two different *information types,* and IE1 and IE2 are *information elements* that have been instantiated from IT1 and IT2, respectively.

The expression "IT1 *aggregates* IT2" means that IT2 is contained in IT1. IE2 is thus part of IE1. The existence of IE2 is dependent on the existence of IE1. See Figure 3 as an example.

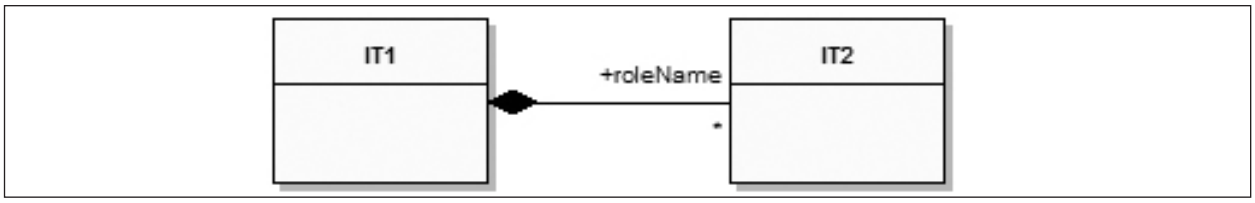


Figure 3: Aggregation of information types

The expression "IT1 generalizes IT2" is equivalent to "IT2 inherits from IT1". This implicates that IT2 receives the attributes, aggregated elements and associations from IT1. Figure 4 illustrates examplarily the generalization concept.

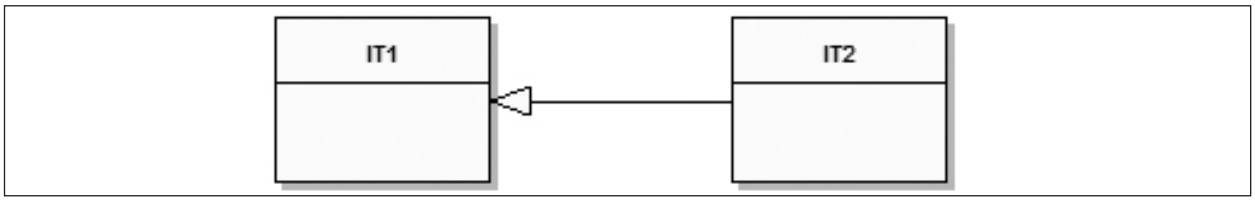


Figure 4: Generaliziation of information types

Associations are always unidirectional. An association from IE1 to IE2 allows the navigation from IE1 to IE2 while the location of IE1 and IE2 is arbitrary. That means, IE1 contains an unambiguous reference to IE2. Note that due to being uni-directional, the reverse navigation from IE2 to IE1 is not possi-ble in this case.

On the information type level, associations are further specified with cardinalities on both origin and target side. The cardinality on the origin side IT1 indicates the possible number of associations. The cardinality on the target side IT2 specifies the possible number of instances of information elements from the associated information type IT2 that can be referenced (see Figure 5).

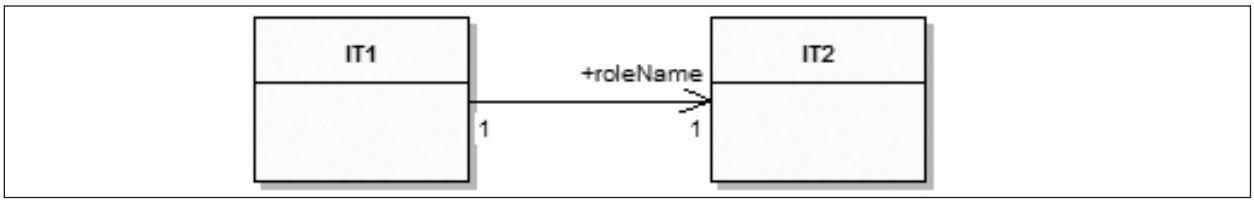


Figure 5: Associations between information types

Associations with an abstract target information type are in fact associations to concrete information types that inherit from the abstract information type.

3.2.1 General structure

Figure 6 shows the information type RIF which is the root of the RIF-model. There must be exactly one instantiation of the root information type RIF per RIF exchange file.

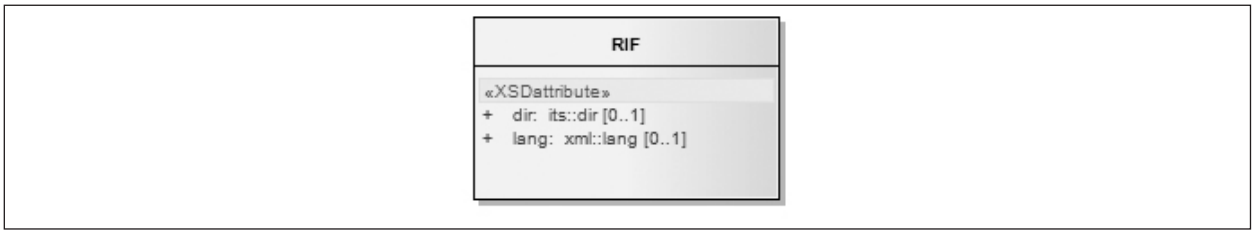


Figure 6: The information type "RIF"

An instantiation of the root type, i.e. a root element, contains the following element attributes:

Attribute in RIF	Description
xml:lang	Language of the contents of the RIF file. The format is defined by the standard for specifying languages in XML documents proposed by the W3C. See http://www.w3.org/TR/xml11/#sec-lang-tag
its:dir	There are non-european languages that use a right-to-left text direction. Thus, the text direction used in the RIF document may be specified here. See the W3C recommendation http://www.w3.org/TR/its/#directionality

A RIF root element aggreagates three parts:

1. The header: instance of RIFHeader information type.
   It stores general metadata information like the time of RIF document creation and the name of the RM tool used as a source.
2. The core content part: instance of RIFContent information type.
   This part contains the actual content from the source RM tool, like requirements, requirement groups and so on.
3. The RM tool specific part: (zero or more) instances of RIFToolExtension information type.
   It can be used to store additional contents from the source RM tool that are not covered by this specification.
   The structure can be defined in a proprietary way, but it must not contain information that can be represented in the core content part (2).

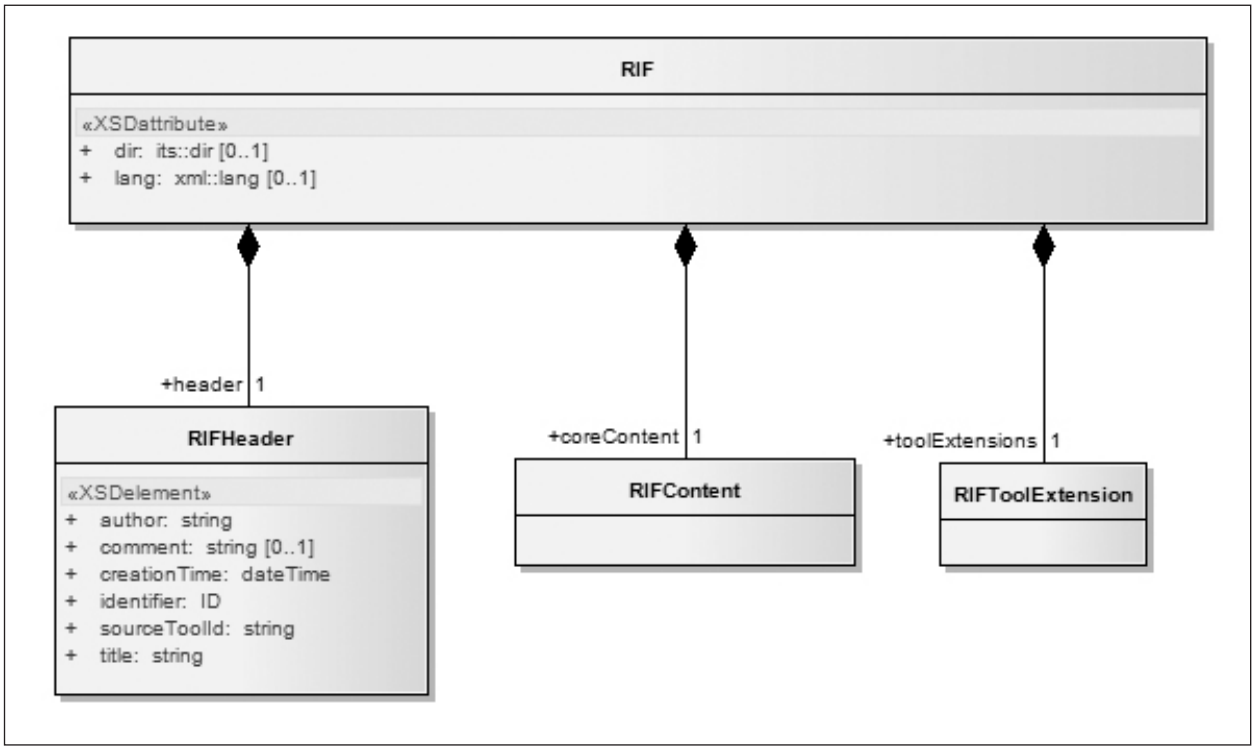


Figure 7: General structure of a RIF document

NOTE: The subelements of RIF are ordered. For more details see chapter 4.

3.2.2 RIFHeader

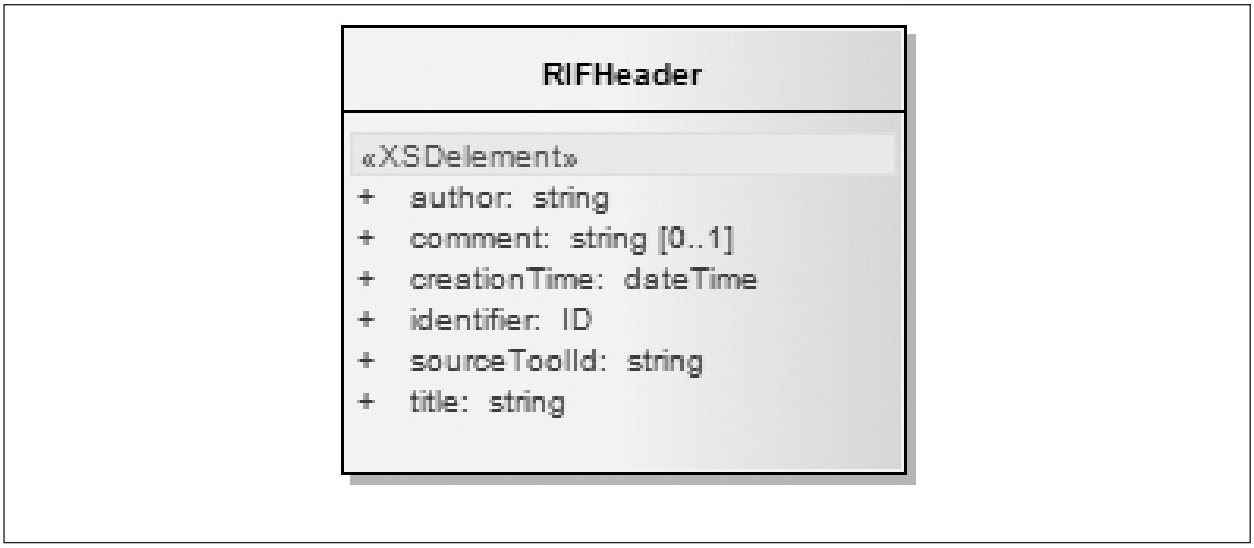


Figure 8: The RIFHeader type

An instantiation of the RIFHeader type contains element attributes that store descriptive information about the interchange:

Attribute in RIF	Description
Identifier	Unique identifier for whole RIF exchange file. The value of the identifier is of the XML Schema data type "xsd::ID"
Title	Title of the RIF exchange file
Comment	Additional comment on the RIF exchange file
Author	Author(s) of the RIF exchange file
creationTime	Time of creation of the RIF exchange file in the format of the XML Schema data type "dateTime" which specifies the time format as CCYY-MM-DDThh:mm:ss with optional time zone indicator as a suffix ±hh:mm. Example: 2005-03-04T10:24:18+01:00 (MET time zone).
sourceToolId	Identifier of the RM tool that has exported the RIF exchange file. Use one of the names defined in row 2 of the RIF Mapping Table or a corresponding one for a new element.

NOTE: the Version attribute that existed in previous RIF specification versions has been removed. The reason for that is: the XML namespace declaration changed in RIF1.2 is sufficient for parsers to differ the RIF specification versions.

3.2.3 RIFContent: Basic information

The RIFContent element includes information elements of information type

- SpecObject,
- SpecGroup,
- SpecHierarchyRoot.
- SpecGroupHierarchyRoot
- DatatypeDefinition,
- SpecType, and
- SpecRelation.

Figure 9 provides an overview of the types of information elements included in a RIFContent element.

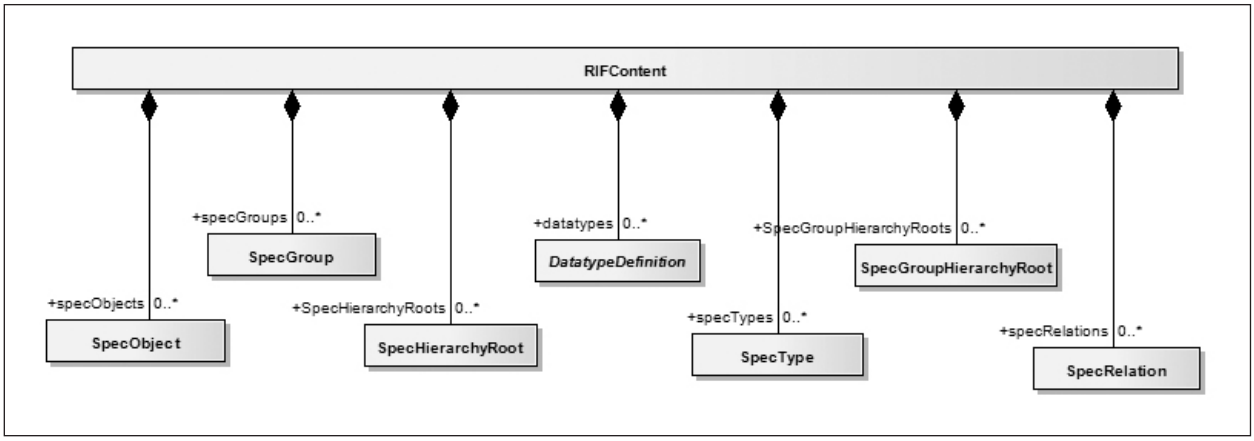


Figure 9: The aggregation relationships of the RIFContent type

The information elements of an information type from the list above occur with an arbitrary number (including zero) inside the core content element.

NOTE: The subelements of RIFContent are ordered. For more details see chapter 4.

The following table gives a short description on the information types that are aggregated by the RIFContent element:

Information type	Short description of information elements of the given type
SpecObject	Constitutes an identifiable requirements object that can be associated with various attributes. This is the smallest granularity by which requirements are referenced. The *SpecObject* instance itself does not carry the requirements text or any other user defined content. This data is stored in *AttributeValue* instances which are associated to the SpecObject instance.
SpecGroup	Defines a group (i.e. a set) of SpecObject instances.
SpecHierarchyRoot	Contains the root node of the tree that hierarchically structures SpecObject instances.
SpecGroupHierarchyRoot	Contains the root node of the tree that hierarchically structures SpecGroup instances.
DatatypeDefinition	Contains definitions of user-defined data types such as integer with a certain range or an enumeration data type. Basic types have also to be defined as instances of *DatatypeDefinition* so that they can be referenced by instances of *AttributeDefinition*.
SpecType	Contains a set of attribute definitions. Any information element that inherits from *ElementWithUserDefinedAttributes* (i.e. instances of *SpecObject, SpecGroup, SpecHierarchyRoot, SpecGroupHierarchyRoot and SpecRelation)* has to be associated with exactly one *SpecType* instance.
SpecRelation	Contains relations (links) between two SpecObject instances.

Before describing the information types from the table above in more detail, the concepts of Identifiable and SpecElementWithUserDefinedAttributes are explained first. These concepts are generic and applied to different information types.

3.2.3.1 Identifiable

Within the RIF exchange file, information elements are able to refer to other information elements using the association relationship. This reference mechanism is realized by generating unique identifiers for each information element that must be referable. These unique identifiers are created by the exporter functionality of the RM tool and are "lifetime identifiers" for the specification elements, i.e. they must not be altered by other tools that import the specification elements and later re-export them. In this way, the tool from which the specification element originates is able to identify the re-imported specification elements that may have been changed.

In the RIF-model, all information elements that must be referable in the way described above inherit from the abstract information type Identifiable which is displayed by Figure 10:

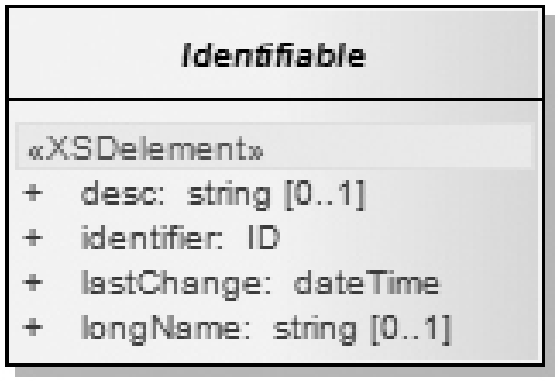


Figure 10: The abstract information type "Identifiable"

Additionally to the identifier, the abstract super-class Identifiable contains other element attributes that contain specific information on an information element. The following Table 1 describes the element attributes:

Attribute in Identifier	Description
desc *(optional)*	Additional description for the information element.
Identifier	The unique identifier of the information element. The value of the identifier has to be of the XML Schema data type "xsd::ID"
lastChange	The date and time of the last change of the information element. This includes the creation of the information element. lastChange is of the XML Schema data type "dateTime" which specifies the time format as *CCYY-MM-DDThh:mm:ss* with optional time zone indicator as a suffix *±hh:mm*. Example: 2005-03-04T10:24:18+01:00 (MET time zone).
longName *(optional)*	The human readable name for the information element. As an example, for *AttributeDefnition* instances, longName contains the column header that usually specifies the meaning of the attribute definition.

Table 1: Attributes of the Identifiable element

Please note that RIF does not prescribe how the unique identifier is generated, as long as it conforms to the XML Schema datatype xsd:ID It may be necessary to transform the RM tool identifier during export from the source RM tool to meet this requirement.

The identifier must not only be unique within the exchange file but also between all RIF exchange files that might exist. This ensures the possibility of uniquely identifying each specification element in situations where many different RM tools are employed to generate requirements in a distributed manner. These requirements can subsequently be collected into one RM tool without running into problems of identification.

3.2.3.2 SpecElementWithUserDefinedAttributes

SpecElementWithUserDefinedAttributes is an abstract information type and is a generalization from the information types SpecObject, SpecGroup, SpecHierarchyRoot and SpecRelation. An UML-based view of this abstract information type is seen in Figure 11.

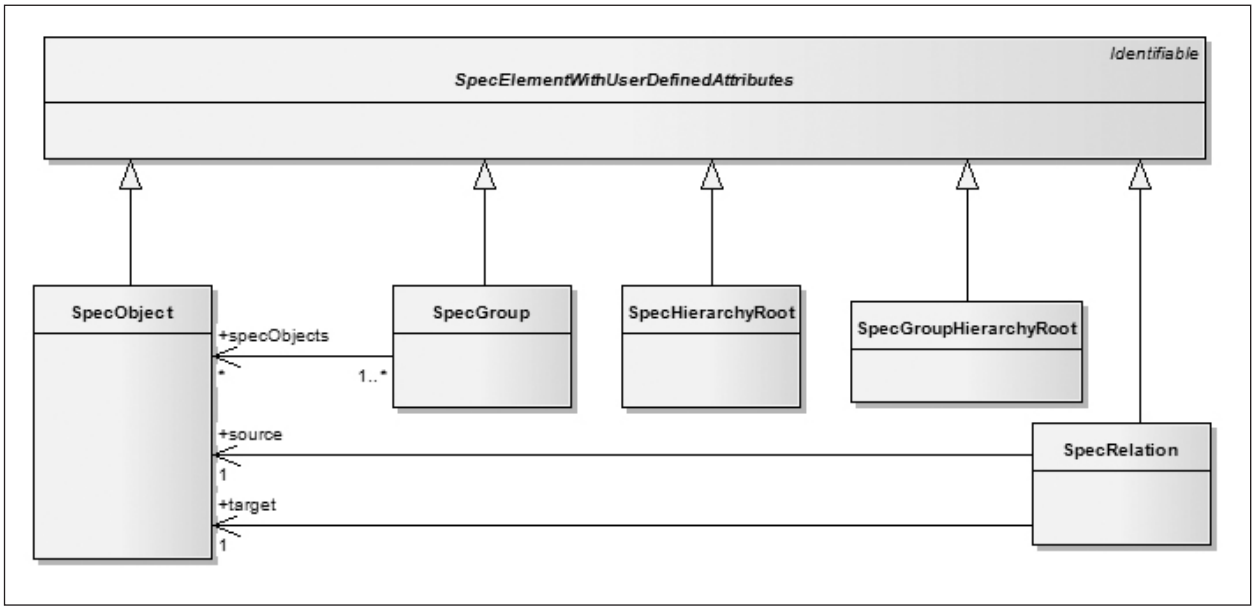


Figure 11: The abstract information type "SpecElementWithUserDefinedAttributes"

The relationships between the four information types that inherit from SpecElementWithUserDefinedAttributes are described later and can be ignored at this point.

From a requirements management point of view, a *specification element* is an information element that carries content relevant for the specification of the system in scope. Thus, not only the requirement itself is a *specification element* but also relations (links) between requirements, the hierarchical structure of requirements and the information to which group a requirement belongs to make up a specification document and are therefore *specification elements*.

For requirements, it is well-known that they can be enriched with additional *requirement attributes* that are defined by the user. In RIF, this concept has been generalized and applied to relations, to the specification group and to the specification hierarchy as well.

Specification elements are usually not associated with arbitrary combinations of attribute elements. For example, specification objects that belong to the same specification group carry the same combination of attribute types. Therefore, it makes sense to define these combinations of attribute types so that they can be easily referred to within the RIF exchange file.

A combination of attribute types is defined by an instance of the information type SpecType. SpecType aggregates the abstract class AttributeDefinition which is a super-class for the different concrete types of attribute definitions (please refer to Section 3.2.4 on data types in the RIF exchange file format). Note that an *attribute definition* can also be seen as the instantiation of an attribute type for a specific specification element. The attribute definition entitles the attribute, associates it with a data type and makes it recognizable so the importing tool knows how to put the attribute values into the different "columns" (in common RM tools, an *attribute definition* for specification objects corresponds to a column carrying user-defined attribute values).

The combination of attribute definitions is specified by the information type SpecType that aggregates an arbitrary number of AttributeDefinitions.

For the information type SpecElementWithUserDefinedAttributes, it is now sufficient to refer to a SpecType (i.e. association with a single SpecType) in order to be associated with a combination of attribute definitions. (For formal reasons, this holds true also if the number of associative attribute definition instances is zero).

At last, SpecElementWithUserDefinedAttributes aggregates an arbitrary number of AttributeValues. Instances of AttributeValues carry concrete values of the type defined by the associated AttributeDefinition. This is further explained in Section 3.2.4 on data types. Figure 12 illusterates the previously described issues.

Please remember that SpecElementWithUserDefinedAttributes is abstract and is thus only used to be inherited from. The association to a SpecType as well as the instances of AttributeValues are therefore located inside the four specification elements that inherit from SpecElementWithUserDefinedAttributes.

Please note also that it is not possible to check on the XML level (e.g. by an XML parser) whether the AttributeValue instances inside a SpecObject instance correctly refer to AttributeDefinition instances that belong to the SpecType instance that is associated with the SpecObject instance. It is also not possible to check on the XML level whether the contents in *AttributeValue* instances really conform to the type definition associated with the attribute. These validity checks have to be performed on a level where this knowledge from the RIF-model is available. In future versions of the RIF-model, these consistency conditions may be specified using OCL (Object Constraint Language) to enable consistency checks on the XML level.

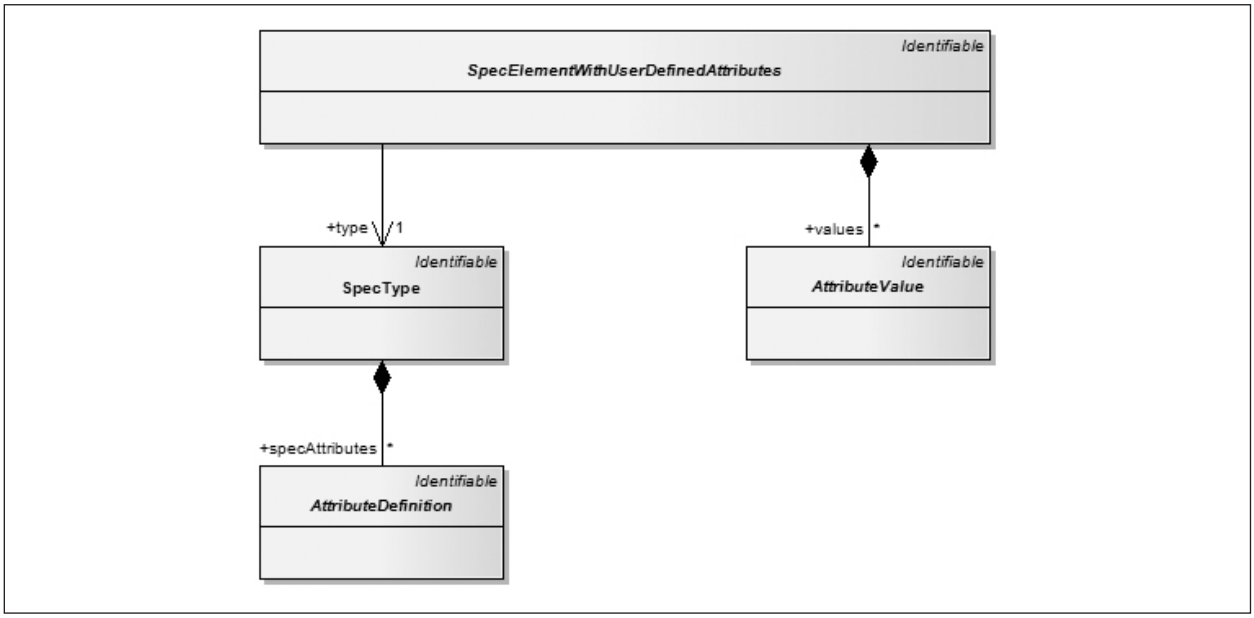


Figure 12: Container-role of "SpecElementWithUserDefinedAttributes"

3.2.3.3 SpecObject

Instances of the information type SpecObject constitute individually identifiable requirements. An instance of SpecObject is "-empty" by itself and therefore contains no data. Data that is associated with a SpecObject instance is stored in instances of AttributeValue which are aggregated by SpecObject (through inheritance from SpecElementWithUserDefinedAttributes). This fact is illusterated by Figure 13.

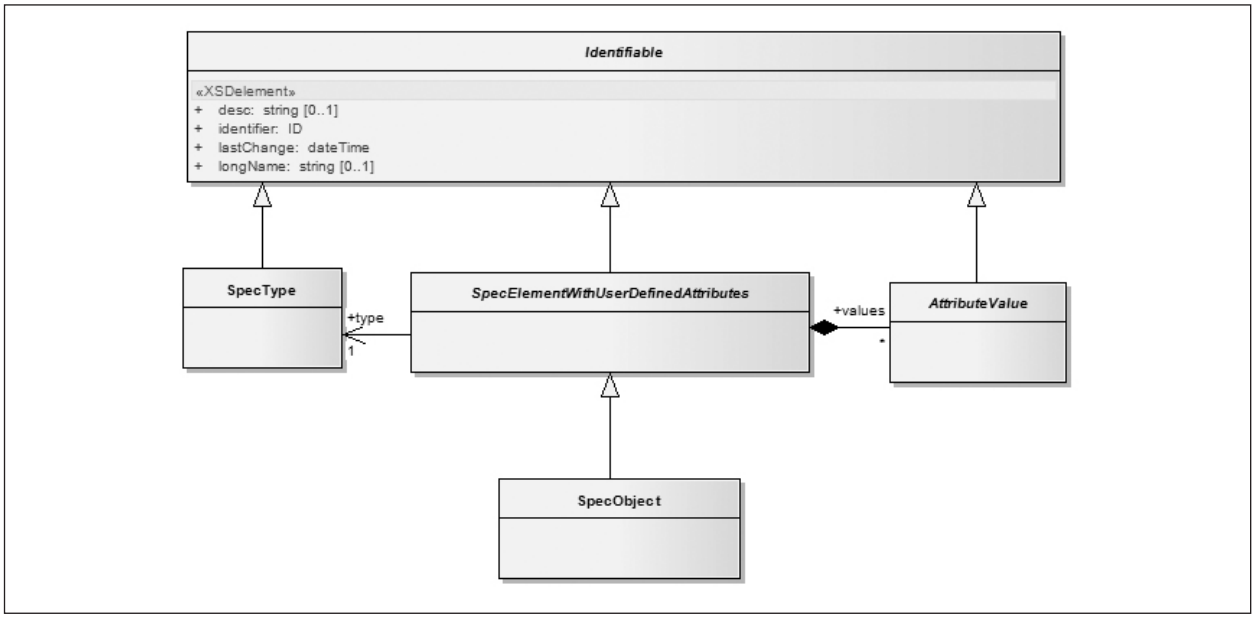


Figure 13: SpecObject derived from the abstract SpecElementWithUserDefinedAttributes

In practical use, there are usually at least two instances of AttributeDefinition that are carried by all SpecObject instances: The requirement identifier from the source tool and the requirement text (i.e. the verbal description of the requirement).

3.2.3.4 SpecGroup

Multiple instances of SpecObject can be grouped together. Such a group is described by an instance of SpecGroup which references instances of SpecObject as displayed in Figure 14.

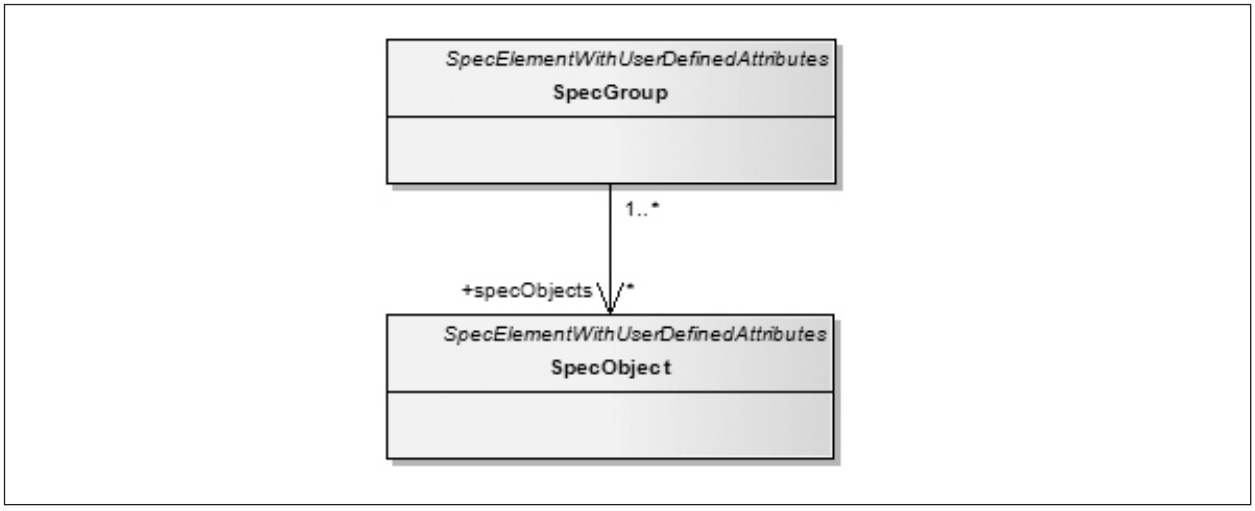


Figure 14: Association between SpecGroup and SpecObject

3.2.3.5 SpecHierarchyRoot

The specification hierarchy describes the hierarchical structure of instances of SpecObject. This hierarchical structure can be seen as a tree with one instance of SpecHierarchyRoot as its root node and an arbitrary number of instances of SpecHierarchy as tree nodes. Links between the nodes of the tree are established by the "recursive" aggregation of instances of SpecHierarchy inside other instances of SpecHierarchy.

The mapping of this tree structure to actual requirements is realized by a reference to a SpecObject instance within each instance of SpecHierarchy. A graphically representation of the SpecHierarchy-element's context is found in Figure 15.

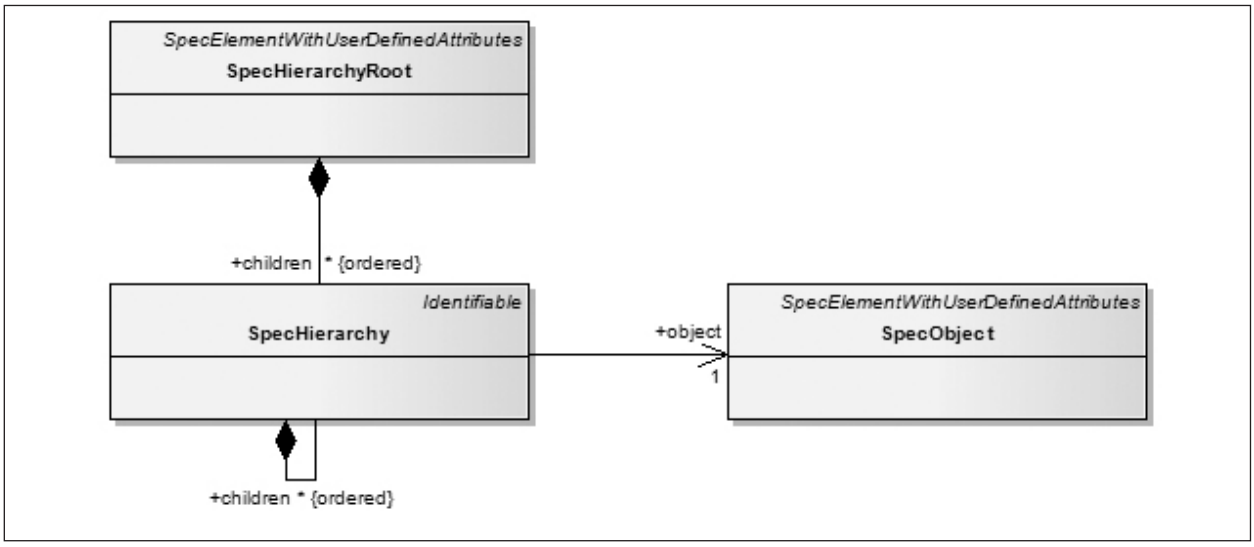


Figure 15: Context of the SpecHierarchy element

3.2.3.6 SpecGroupHierarchyRoot

Analogous to the information type SpecHierarchyRoot, a hierarchy of SpecGroups can be created by using a SpecGroupHierarchyRoot element and aggregated SpecGroupHierarchy elements (see Figure 16).

In contrast to SpecHierarchy elements, SpecGroupHierarchy elements must not be used to represent the hierarchical structure of requirements.

Instead, SpecGroupHierarchy elements may be used to represent the internal structure of the RM tool repository. This may for instance be a structure of nested packages or projects containing requirements (but not the requirements themselves).

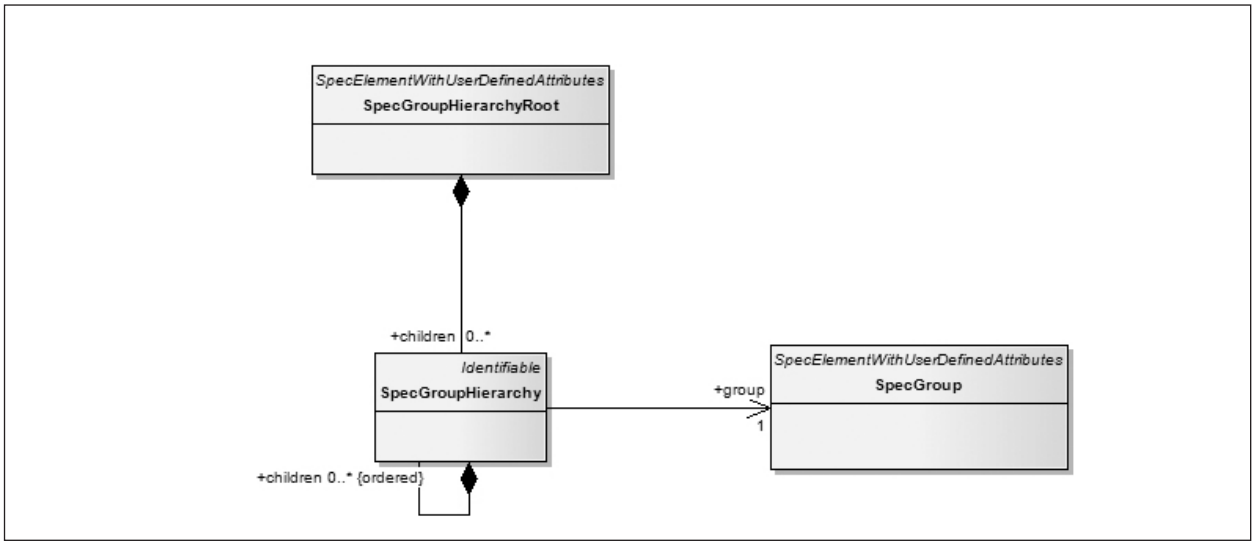


Figure 16: Context of the SpecGroupHierarchy element

NOTE: The import tool can decide how to handle the SpecGroupHierarchy information for import. For example, it can be inacceptable to update a package hierarchy in the target RM tool based on the structure in the source tool.

3.2.3.7 DatatypeDefinition

In RIF, there are three kinds of data types:

1. Simple data types (i.e. Integer, Real, Boolean, String)
2. Enumeration data types (both single and multiple enumerations)
3. Complex data types

Complex data types are in fact data files that can be either binary or have an XML based content. Although there are partially large differences in terms of data concepts between these kinds of data types, a common modeling approach has been realized as far as reasonable.

The abstract super-class for the three kinds of data types is DatatypeDefinition. The abstract classes of data types and their hierarchy are displayed in Figure 17.

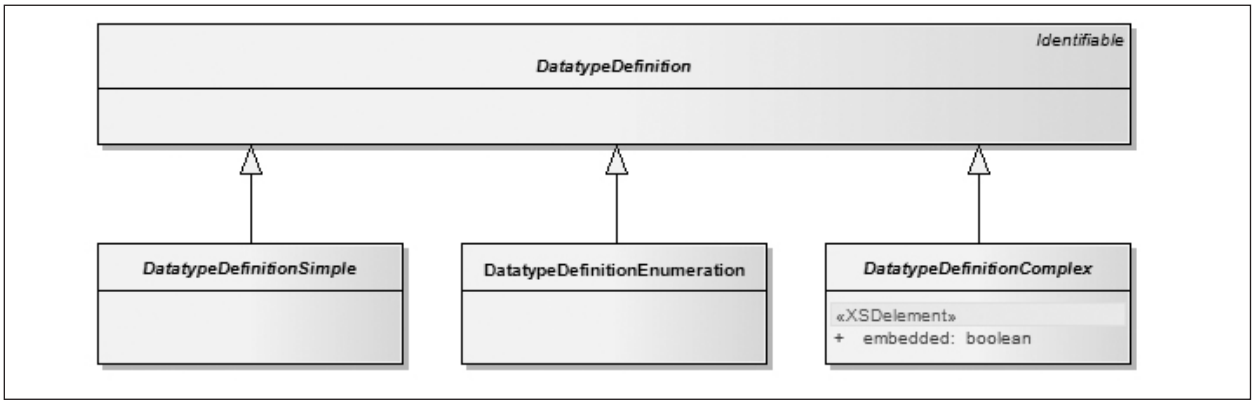


Figure 17: The abstract classes of data-types

Concrete information types with data type definitions inherit from the abstract information types DatatypeDefinitionSimple, DatatypeDefinitionEnumeration *or* DatatypeDefinitionComplex. They are described in more detail in Section 3.2.4.

A concrete information type defining a data type is referenced by an instance of AttributeDefinition. An AttributeDefinition is thus typed by this association.

3.2.3.8 SpecType

An instance of SpecType defines a combination of AttributeDefinition instances. This is realized by aggregating an arbitrary number of instances of AttributeDefinition within a SpecType instance and illustrated in Figure 18.

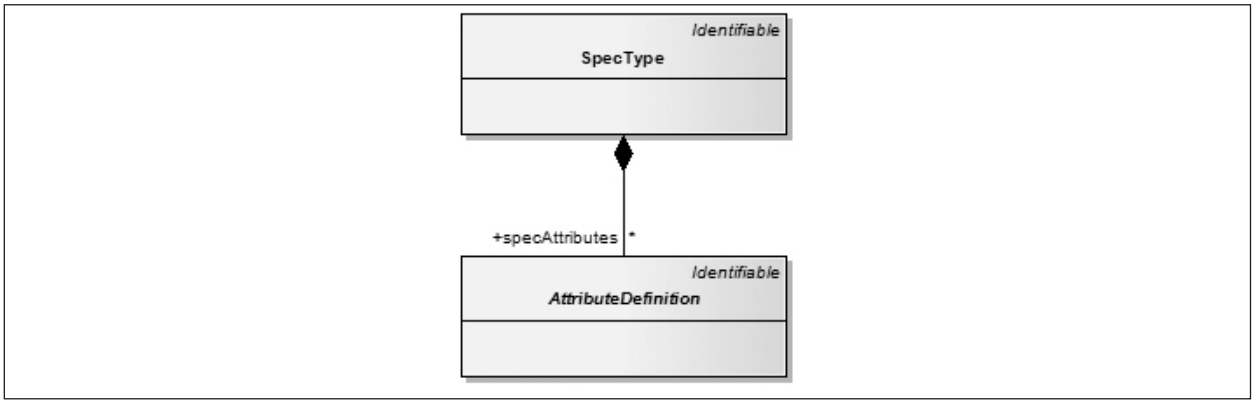


Figure 18: Aggregation of AttributeDefinition by SpecType

SpecType instances are referenced by instances of specification elements (i.e. instances of SpecObject, SpecGroup, SpecHierarchyRoot and SpecRelation). In this way, a combination of attribute definitions is associated with a specification element.

The information type AttributeDefinition itself is an abstract super-class for three concrete attribute definitions based on each of the three kinds of data types. This hierarchy is displayed in Figure 19.

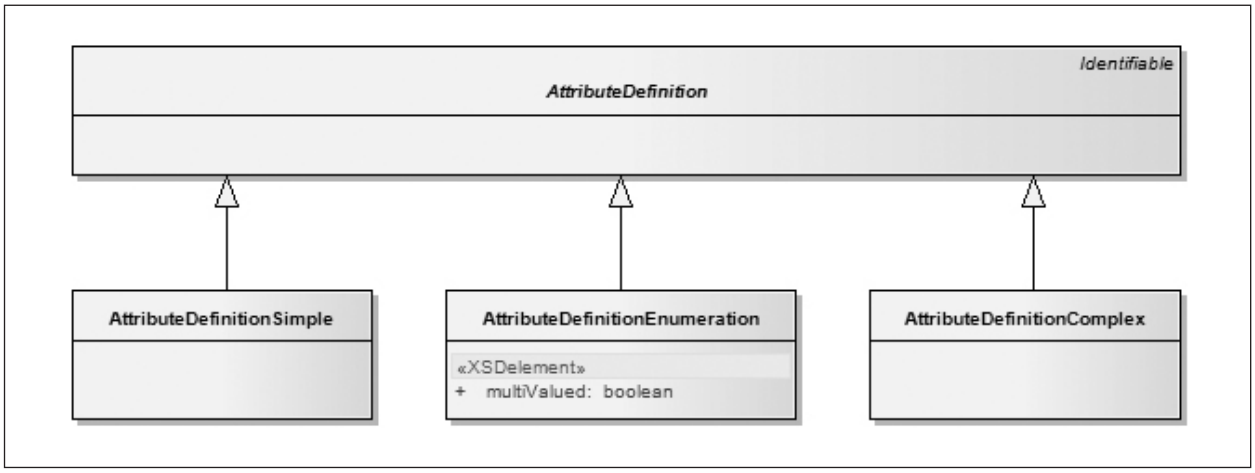


Figure 19: The AttributeDefinition class hierarchy

3.2.3.9 SpecRelation

An instance of SpecRelation indicates a relation between two instances of SpecObject as indicated by Figure 20.

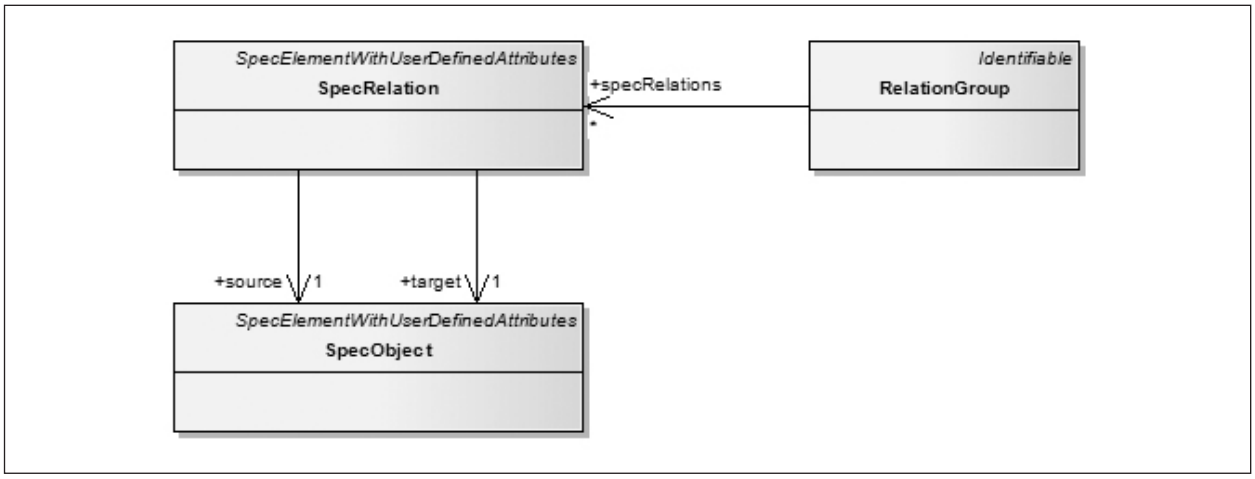


Figure 20: UML diagram of SpecRelation

This relation is a uni-directional reference realized by the associations of a SpecRelation instance with the role names "source" and "target" to two distinct instances of SpecObject.

Instances of SpecRelation are themselves grouped together to form types of relations (e.g. "realizes", "is associated with" etc.). Relation groups are also used for access policies. Groups of SpecRelation instances are formed by an instance of RelationGroup which has an association to one or multiple instances of SpecRelation.

3.2.4 RIFContent: Data types

As already mentioned, there are three basic kinds of data types in RIF:

1. Simple data types
2. Enumeration data types
3. Complex data types

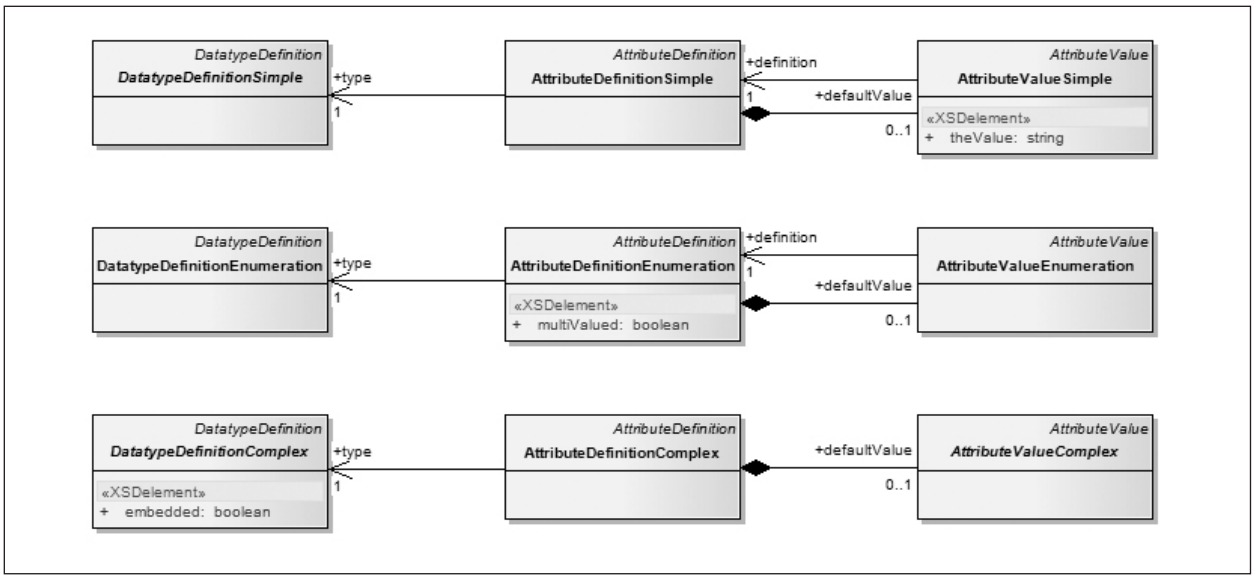


Figure 21: The RIF data types and their relations

Each of these three data type categories has a representation as

- data type definition,
- attribute definition and
- attribute value.

For the data types, the attribute definition is typed by a data type definition through the respective association. Concrete instances of attribute values have a reference to their respective attribute defini-tion. Furthermore, an attribute definition can have a default value and thus may aggregate an attribute value instance of respective type. The relations between DatatypeDefinition, AttributeDefinition- and AttributeValue-elements are illustrated in Figure 21.

3.2.4.1 Simple data types

A simple data type is one of Integer, Real, Boolean or String. The concrete information types DatatypeDefinitionDate, DatatypeDefinitionInteger, DatatypeDefinitionReal, DatatypeDefinitionBoolean and DatatypeDefinitionString therefore inherit from the abstract super-class DatatypeDefinitionSimple. The class-hierarchy is shown in Figure 22.

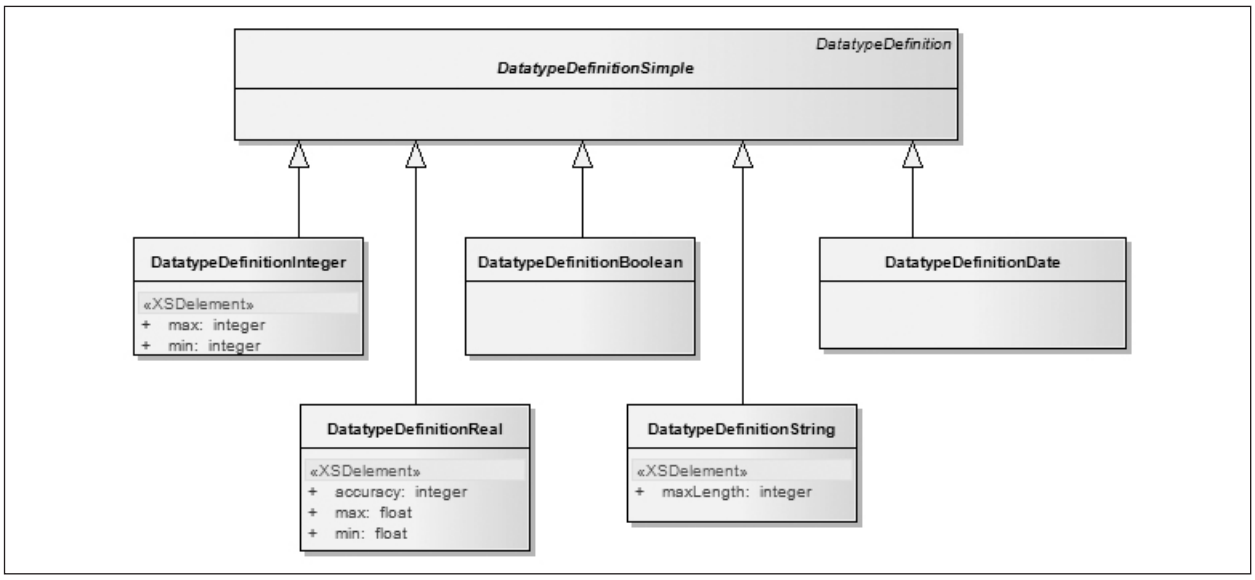


Figure 22: Class-hierarchy of DataTypeDefinitionSimple

Instances of the concrete data type definitions are further specified by additional element attributes (e.g. "min" and "max" to indicate ranges).

Concrete values of a simple data type are stored within an instance of AttributeValueSimple as an element attribute ("theValue") of type String which is illustrated in Figure 23. This way, the modeling of data types and attribute values has been simplified but on the other side, no data type checking for attribute values can be performed on the XML-Schema level. This disadvantage is acceptable because the import functionality of an RM-tool has to perform data type checking for all data type categories anyway.

For encoding the values with simple datatypes, see the following table:

Simple Datatype	Encoding of theValue defined in (W3C recommendation)
DatatypeDefinitionBoolean	http://www.w3.org/TR/xmlschema-2/#boolean
DatatypeDefinitionReal	http://www.w3.org/TR/xmlschema-2/#double
DatatypeDefinitionInteger	http://www.w3.org/TR/xmlschema-2/#integer
DatatypeDefinitionString	http://www.w3.org/TR/xmlschema-2/#string
DatatypeDefinitionDate	http://www.w3.org/TR/xmlschema-2/#isoformats

For example: an instance of AttributeValueSimple is (transitively) associated with an instance of DatatypeDefinitionBoolean. In this case, the lexical content of its theValue attribute shall be one of the following: 0, 1, true or false.

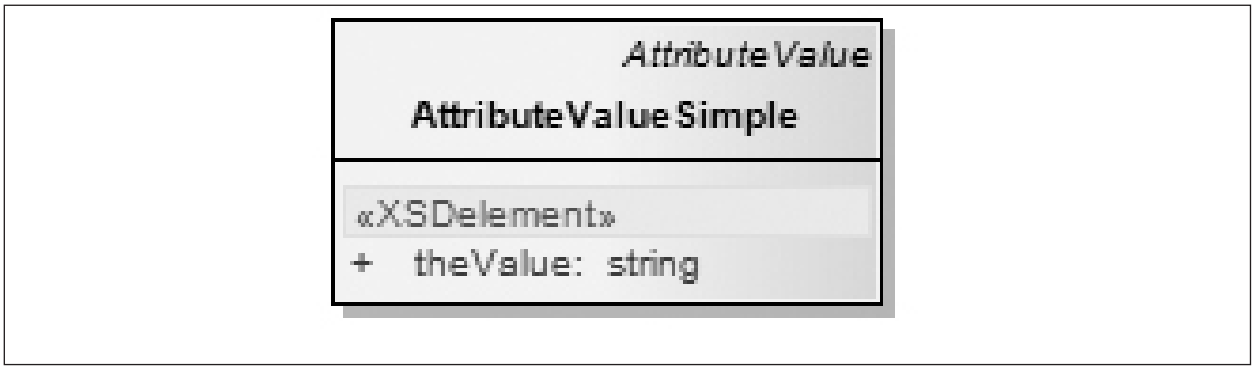


Figure 23: The AttributeValueSimple class

3.2.4.2 Enumeration data types

RIF supports both single enumeration and multi-value enumeration data types. The enumeration type is indicated by the element attribute "multiValued" inside an information element of type AttributeDefinitionEnumeration.

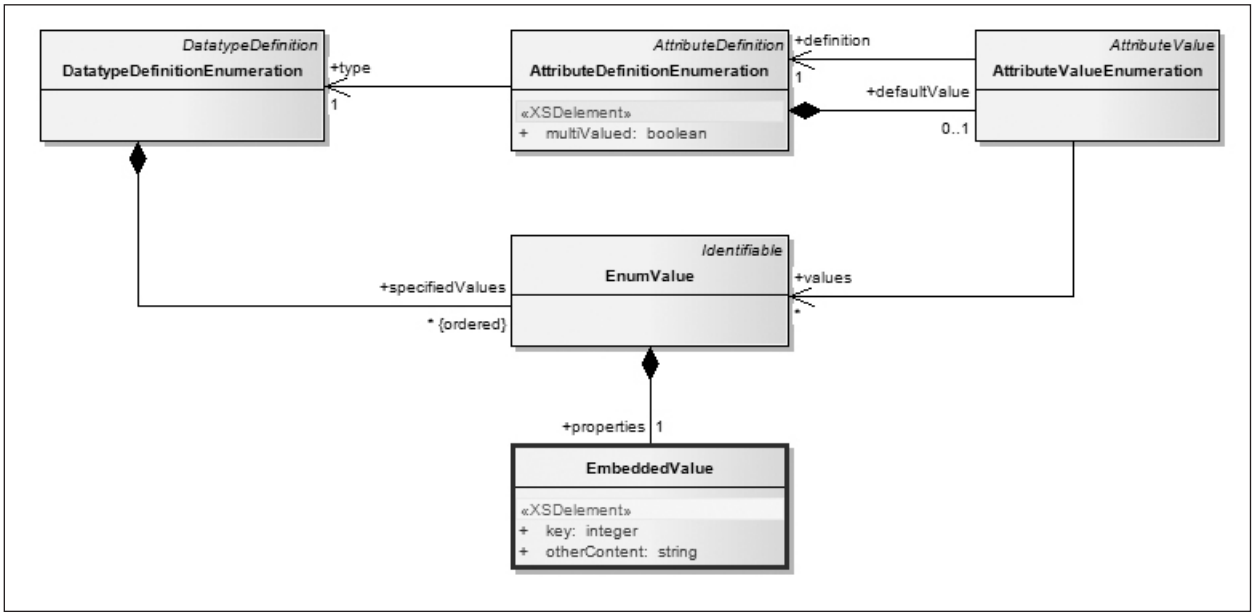


Figure 24: The AttributeDefintionEnumeration element and it's relations

Both single and multiple enumerations types are based on the same set of enumeration values which are instances of EnumValue. An enumeration data type is made up of a set of enumeration values. Therefore, an instance of DatatypeDefinitionEnumeration aggregates an arbitrary number of EnumValue instances.

Concrete enumeration values are stored in an instance of AttributeValueEnumeration as associations to the respective EnumValue instances. An AttributeValueEnumeration instance is defined by an instance of AttributeDefinitionEnumeration which also indicates whether the enumeration type is "single" or "multi-value". Within an instance of AttributeDefinitionEnumeration, the set of possible enumeration values is defined by the reference to a DatatypeDefinitionEnumeration instance. Furthermore, an instance of AttributeDefinitionEnumeration can have a default value which is indicated by the aggregation of an AttributeValueEnumeration instance. The relations of the AttributeDefinitionEnumeration class are displayed in Figure 24.

Information elements of type EnumValue aggregate EmbeddedValues instances which contain additional data for the enumeration value (e.g. a key). Please note that the information type EmbeddedValues is likely to be extended in a future version of RIF.

3.2.4.3 Complex data types

Complex data types contain in principle arbitrary structured data. With the exception of XHTML-based data, the deeper meaning and interpretation of complex data is not apparent for the RIF importer and exporter functionality. Complex data is handled on the application level and therefore contains additional information (e.g. file format, associated application type) to enable the intended processing of the complex data.

RIF distinguishes three types of complex data:

- Document data (i.e. XHTML based data)
- XML based data
- Binary data

Although XHTML is in fact a type of XML data, RIF treats XHTML as a complex data type of its own since certain functionality of RIF (e.g. text formatting, object embedding) is realized through XHTML with rules on the use of specific XML elements of XHTML (see Section 3.2.6 for further details).

XML based data is "typed" by an XML-Schema and is embedded within the RIF exchange file in order to simplify its handling.

Binary data can be any data, and thus the structure of binary data is not known to RIF. Binary data is either embedded in an encoded format (e.g. BASE64) within the RIF exchange file or it is stored externally and referenced from within the RIF exchange file. The relations of the three complex data types in RIF are displayed in Figure 25.

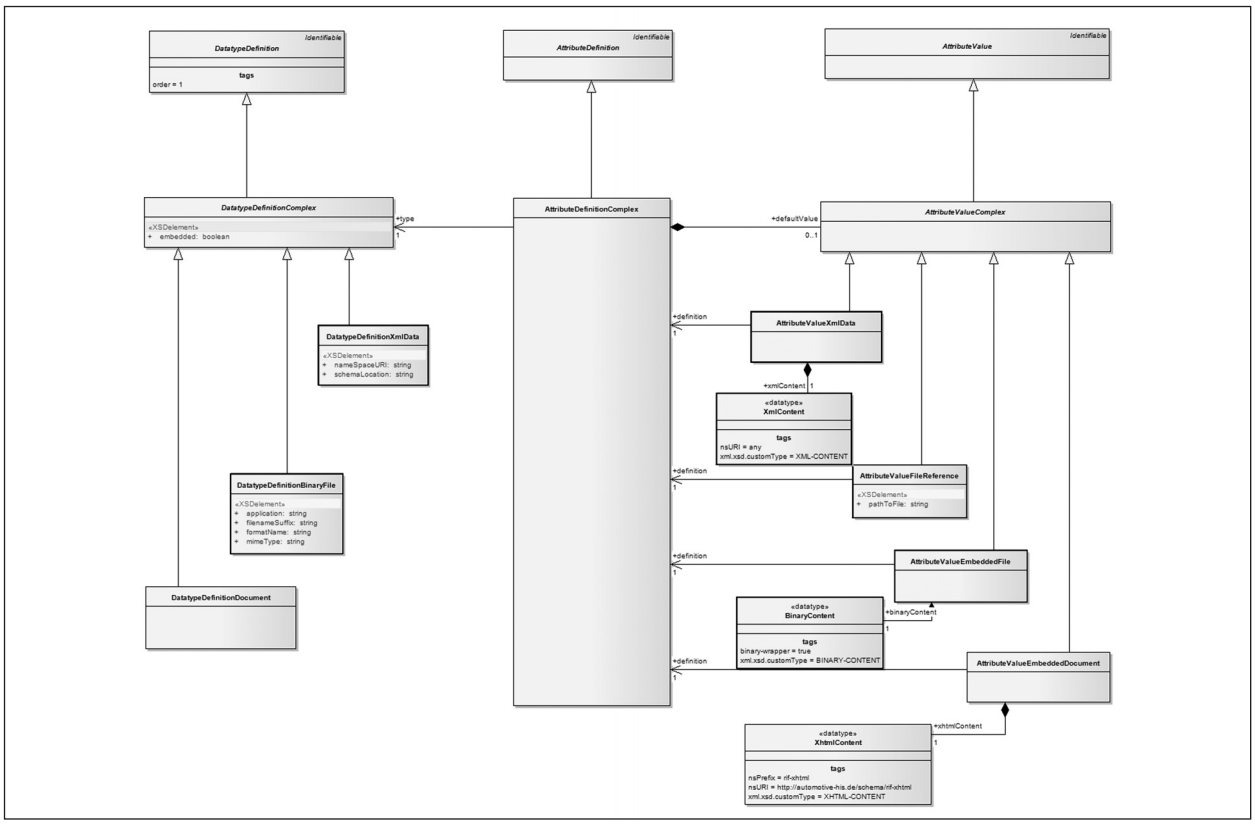


Figure 25: Illustration of the relations of complex data in RIF

A complex data type can be either document data, XML data or a binary file. Thus, the information types DatatypeDefinitionDocument, DatatypeDefinitionXmlData and DatatypeDefinitionBinaryFile inherit from the abstract super-class DatatypeDefinitionComplex. Instances of these information types have element attributes that further characterize the complex data type:

- For XML based data types, the reference to the XML-Schema has to be specified.
- For binary data, the format name, the MIME type, the file name suffix and the associated application type has to be specified.

Finally, complex attribute values are instantiated from an information type of either AttributeValueEmbeddedDocument, AttributeValueXmlData, AttributeValueFileReference or AttributeValueEmbeddedFile depending on the type (document, XML or binary) and location (embedded or external) of the complex data.

Please note that instances of AttributeValueDocument, AttributeValueXmlData and AttributeValueEmbeddedFile are in principle wrappers for an embedded document, XML or binary content, respectively. This issue is stated in Figure 26. The storage of these contents is out of scope of the RIF model. This is indicated to the conversion scripts using the aggregated information types "XhtmlContent", "XmlContent" and "BinaryContent" which have specific tags. Please refer to Section 4.1.5 for more details on this issue.

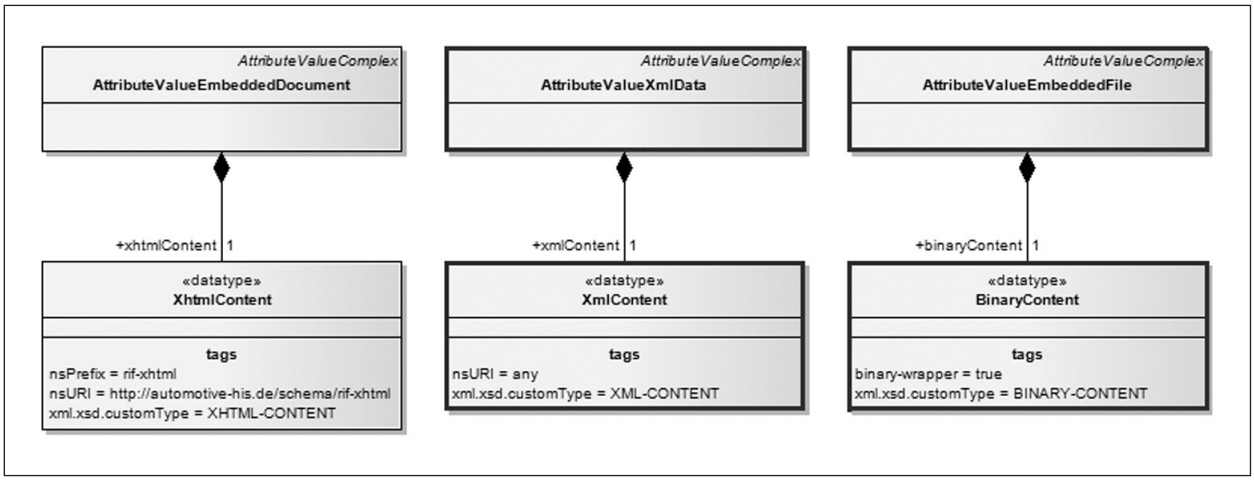


Figure 26: The wrappers for the complex data types

3.2.5 RIFContent: Access policies

RIF supports access policies that are valid for the receiver of the RIF exchange file. In RIF, many types of information elements can be associated with a definition of access rights (i.e. an instance of AccessPolicy). Please note that in the diagram below, Figure 27, the relationships among the information types that are referenced by AccessPolicy are hidden due to simplification reasons.

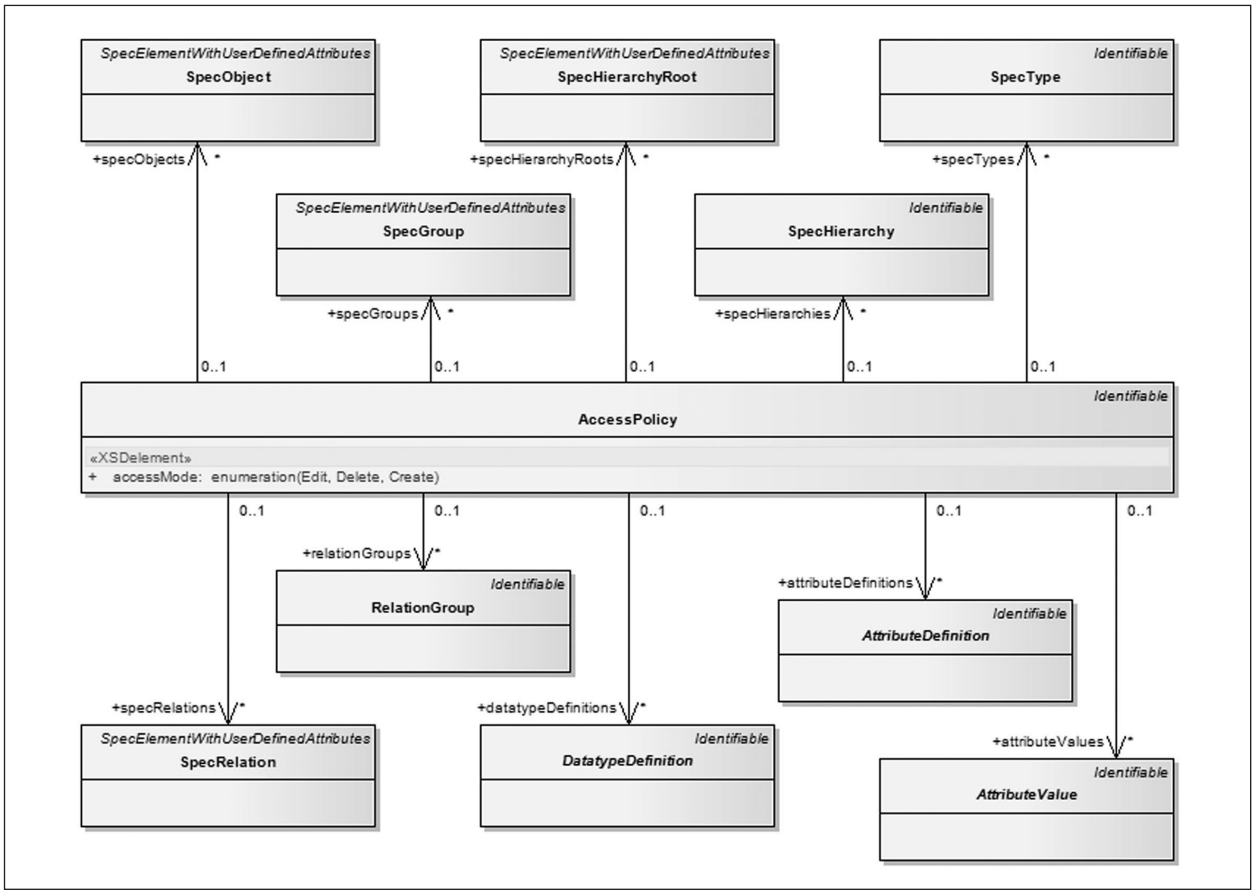


Figure 27: Overview of class-references of AccessPolicy

Dependent on the type of the information element, the associated access policy that is defined as alternatively "Edit", "Delete" or "Create" have different interpretations and semantics.

A generic rule for access policies is that they are inherited along aggregations but **not** along associa-tions. In this way, an access policy is also valid for the information elements that are located within the information element that is associated with the access policy.

This inherited access policy is valid unless a different access policy overrides it. (This concept is equivalent to the scoping rules of local and global variables in modern programming languages.)

The following Table 2 specifies the application of the access policy for the relevant information types:

Information type	Application of the associated access policy	Access policy is inherited by
SpecObject	• Renaming (longname) of the SpecObject instance • Changing of the associated SpecType instance	• AttributeValue instances for SpecObject'
SpecGroup	• Renaming (longname) of the SpecGroup instance • Changing of the associated SpecType instance • Adding/deleting of SpecObjects instances to/ from the SpecGroup instance	• AttributeValue instances for SpecGroup • RelationGroup instances
SpecHierarchyRoot	• Renaming (longname) of the hierarchy structure • Modification of headers of the main chapters (1st level instances of SpecHierarchy)	• AttributeValues instances for SpecHierarchyRoot • 1st level instances of SpecHierarchy
SpecHierarchy	• Restructuring, adding and deleting of sub-chapters within the chapter for which the access policy is defined	• SpecHierarchy instances (recursively)
SpecType	• Renaming (longname) of the SpecType instance • Adding and deleting of AttributeDefinition instance to/ from the SpecType instance	• AttributeDefinition instances
SpecRelation	• Renaming (longname) of the SpecRelation instance • Changing of the associated SpecType instance • Changing of the source and target associations to SpecObject instances	• AttributeValue instances for SpecRelation
RelationGroup	• Renaming (longname) of the RelationGroup instance • Changing of the associated SpecType instance • Adding/deleting of SpecRelation instances to/ from the RelationGroup instance	
DatatypeDefinition	• Renaming (longname) of the DatatypeDefinition instance • Modification of the DatatypeDefinition instance	
AttributeDefinition	• Renaming (longname) of the AttributeDefinition instance • Changing the reference to the associated DatatypeDefinition instance	
AttributeValue	• Modification of the content of the AttributeValue instance	

Table 2: Rules of the information types, associated access policy and it's inheritance

3.2.6 RIFContent: XHTML name space

As already mentioned previously, XHTML is a complex data type on its own with rules on the use of certain XHTML elements. The term "XHTML namespace" is used here because for data based on the complex XHTML data type, the namespace within the exchange file is temporarily switched from the RIF namespace to the XHTML namespace. This allows validating against different XML-Schemas (i.e. against the RIF-Schema and against an XHTML-Schema) resulting in more independency between the different XML-Schemas.

For RIF, the XHTML namespace aligns to the XHTML 1.0 strict specification and thus supports tables and various text-/font-styles without the need of CSS. The original XHTML-strict-schema was slightly changed to meet the RIF requirements. These changes include removing unnecessary tags and adding two tags for underline and strike-through. The following Table 3 lists all tags currently available in the RIF-XHTML namespace.

Tag	Comments
Text elements	
 	Forced line break
	Generic language/style container
<object>	Generic embedded object
<img>	Embedded image
<tt>	Teletype or monospaced text style
<i>	Italic text style
<b>	Bold text style
<big>	Large text style
<small>	Small text style
<u>	Underlined text style
<strike>	Strike-through text
	Indicates emphasis
	Indicates stronger emphasis
<dfn>	Indicates that this is the defining instance of the enclosed term
<code>	Designates a fragment of computer code
<q>	Short inline quotation
<samp>	Designates sample output from programs, scripts, etc.
<kbd>	Indicates text to be entered by the user
<var>	Indicates an instance of a variable or program argument
<cite>	Contains a citation or a reference to other sources
<abbr>	Indicates an abbreviated form
<acronym>	Indicates an acronym
<sub>	Subscript
<sup>	Superscript
<ins>	Inserted text
<del>	Deleted text
<a>	Anchor
Block level elements	
<h1>	Heading
<h2>	Heading
<h3>	Heading
<h4>	Heading
<h5>	Heading

Tag	Comments
<h6>	Heading
	Unordered list
	Ordered list
<dl>	Definition list
<pre>	Preformatted text
<hr>	Horizontal rule
<blockquote>	Long quotation
<address>	Information on author
<p>	Paragraph
	Generic language/style container
<table>	Used to describe tables
Document structure	
<html>	Document root element
<body>	Document body
Miscellaneous	
	List item
<dt>	Definition term
<dd>	Definition description
<param>	Named property value
<caption>	Table caption
<thead>	Table header
<tbody>	Table body
<tfoot>	Table footer
<colgroup>	Table column group
<col>	Table column
<tr>	Table row
<th>	Table header cell
<td>	Table data cell

Table 3: Tags used by the RIF-XHTML schema

The tags define logical mark-ups for the text and not directly its physical presentation. For the physical presentation, the common implementation (e.g. as realized in Microsoft's Internet Explorer) must be adopted.

3.2.6.1 Object embedding in the XHTML namespace

With RIF, it is also possible to embed arbitrary objects (e.g. pictures) within text. This functionality is based on Microsoft's clipboard and OLE (Object Linking and Embedding) technology and is realized within the RIF-XHTML namespace.

The following Table 4 gives an overview of the XML-elements and -attributes of the RIF-XHTML object that are used for embedding objects:

Element	Attributes	Attribute types	Minimal Content Model
Object	Classid	URI	(PCDATA | Flow | param)*
	data	URI	
	height	Length	
	name	CDATA	
	type	ContentType	
	width	Length	

Table 4: XML-elements and -attributes of the RIF-XHTML object

Please refer to Section 4.3 for more details on embedding objects.

3.2.7 RIFToolExtension

The RM tool specific part may be used to aggregate RM tool content not covered by this RIF specification. For example: it may be used to represent instances of the concept of Views found in RM tools.

The XML elements in the XML representation of the RM tool specific content should be namespace qualified. If an XML namespace is used, it must differ from the XML namespace of the RIF information type as defined by the targetNamespace of the RIF XML Schema.

The concepts represented by the information types of the core content part should not be reused or altered for the tool extensions part. For example: content of requirement attributes should not be specified in the tool extensions part, as the concept of requirement attributes is already covered by the core content part.

Instead, extensions to the concepts of the core content part should be represented like this:

1. Place instances of existing RIF information types in the core content part.
2. Place extensions in the tool extensions part and reference the extended RIF elements (by using an XML attribute or XML element of type RIF:REF).

4 Implementation Details for RIF

This chapter gives detailed information that is relevant for the implementation of RIF. It covers how the XML structure of RIF exchange files is actually derived from the RIF-information -model, how additional data objects are embedded and how the RIF exchange file together with external files is to be packaged.

4.1 XML Formalization

The RIF-model has been presented in Chapter 3 on an abstract level using UML class diagrams. This chapter contains the rules on the XML formalization for the RIF-model. These rules are automatically applied when converting the RIF-model into an XML-Schema which describes valid RIF exchange files.

In order to keep this part of the documentation as simple as possible, the rules for the XML formalization are illustrated on the level of an XML-based RIF exchange file. Rules and examples for the conversion of the RIF-model into an XML-Schema are not given because for tool developers, it is more important to know how the RIF exchange file must be generated and structured. Furthermore, the XML- Schema itself that is generated from the RIF-model can be deduced from the rules and examples given on the XML level.

4.1.1 General rules

Each concrete information type in the RIF-model is mapped onto an XML element. The name of the XML element is constructed by converting the information type's name into uppercase letters with additional hyphens ("-") indicating word separations that have originally been indicated by uppercase letters or by a numeric character inside the name.
Thus, by definition, a "word" is one of the following:

- First letter is upper-case followed by lower-case letters.
- All letters are upper-case.
- Contiguous numeric characters

For example, the name TestECUClass12ADC is converted into an XML element with name TEST-ECU-CLASS-12-ADC.

Example:

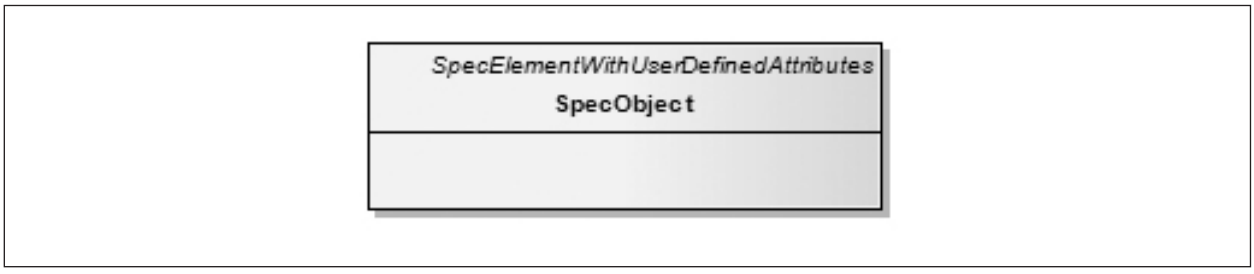


Figure 28: Example of the notation of the name of a modelled RIF element

The information type SpecObject (see Figure 28) is mapped onto an XML element with name *"SPEC-OBJECT":*

```
<SPEC-OBJECT>
…
</SPEC-OBJECT>
```

Element attributes that are defined within an information type are also mapped onto XML elements with the same rule for name conversion.

Example:

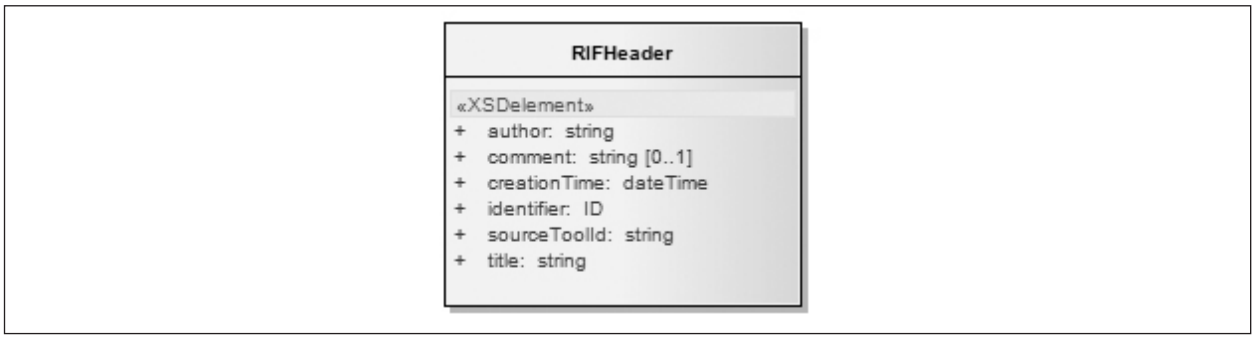


Figure 29: Example RIFHeader element with attributes

The information type RIFHeader (see Figure 29) is mapped onto an XML element "RIF-HEADER" with sub-elements representing the element attributes. The concrete values for the element attributes are stored inside these nested elements:

```
<RIF-HEADER>
    <AUTHOR>…</AUTHOR>
    <COMMENT>…</COMMENT>
    <CREATION-TIME>…</CREATION-TIME>
    <IDENTIFIER>…</IDENTIFIER>
    <SOURCE-TOOL-ID>…</SOURCE-TOOL-ID>
    <TITLE>…</TITLE>
</RIF-HEADER>
```

In the RIF-model, each element attribute has a data type. These data types are based on the XML data types.

The default multiplicity for an information type attribute is 1. Optionally, a different multiplicity can be specified by adding *[LowerLimit .. UpperLimit]* behind the data type. For example, *"desc: String [0..1]"* denotes an information type attribute with name "desc" and of type "String" which can occur zero or one time.

4.1.2 Inheritance

In the RIF model, SpecObject inherits from the abstract information type SpecElementWithUserDefinedAttributes which itself inherits from Identifiable as shown in Figure 30:

Example:

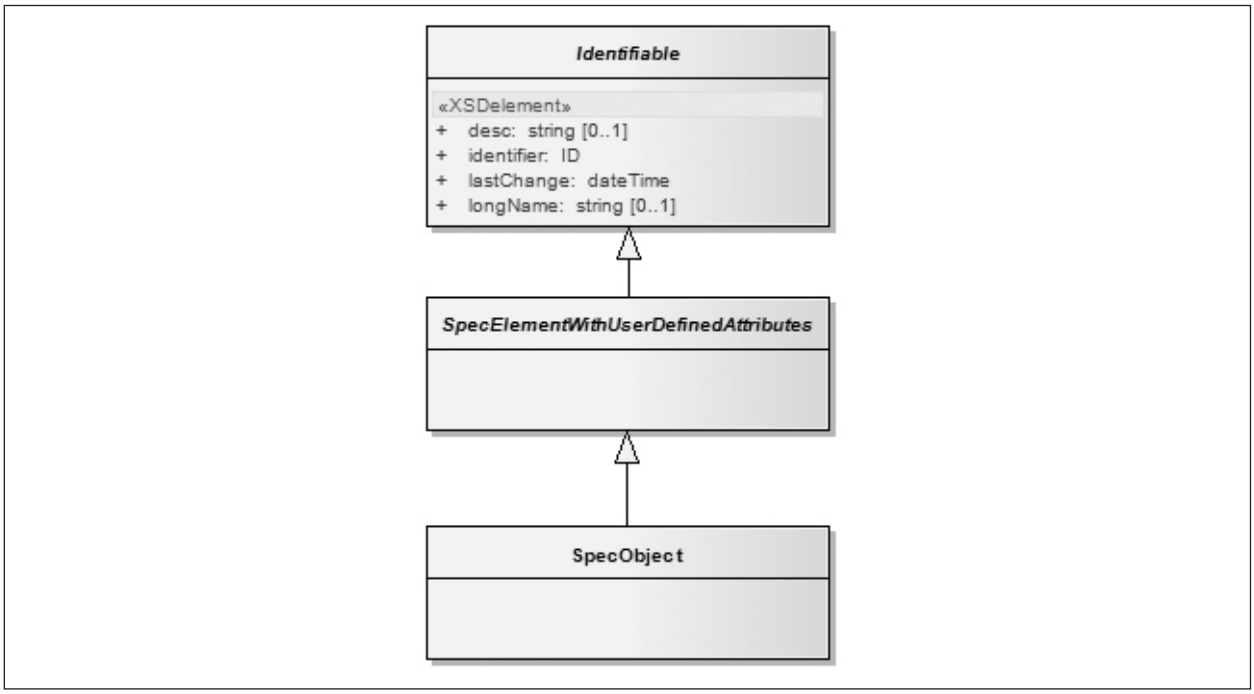


Figure 30: Example of inheritance

A shortcut notation for inheritance is to write the name of the information type from which is inherited with italic characters in the upper right corner of the box as already depicted in Section 4.1.1.

In general, an information type A that inherits from information type B receives all element attributes that are defined inside information type B independent whether B is abstract or concrete. The example above thus results in the following XML data:

```
<SPEC-OBJECT>
        <IDENTIFIER> … </IDENTIFIER>
        <LONG-NAME> … </LONG-NAME>
        <DESC> … </DESC>
        <LAST-CHANGE> … </LAST-CHANGE>
        …
</SPEC-OBJECT>
```

4.1.3 Composition

Aggregated information types are mapped onto nested XML elements. If the multiplicity of the aggregated information element is 1, the composition works in the same way as element attributes.

Example:

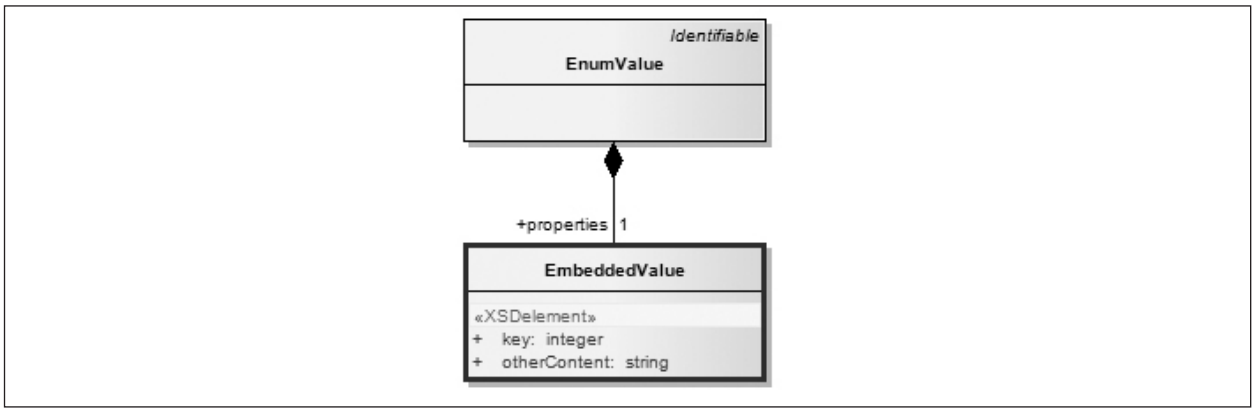


Figure 31: Example of a composition

The information type EmbeddedValue is part of the information type EnumValue (see Figure 31):

```
<ENUM-VALUE>
        …
        <PROPERTIES>
                <EMBEDDED-VALUE>
                        <KEY> … </KEY>
                        <OTHER-CONTENT> … </OTHER-CONTENT>
                </EMBEDDED-VALUE>
        </PROPERTIES>
        …
</ENUM-VALUE>
```

For aggregation of multiple instances of an information type, the nested XML elements occur in sequence within the wrapping XML element that is made up of the source role name of the composition.

Example:

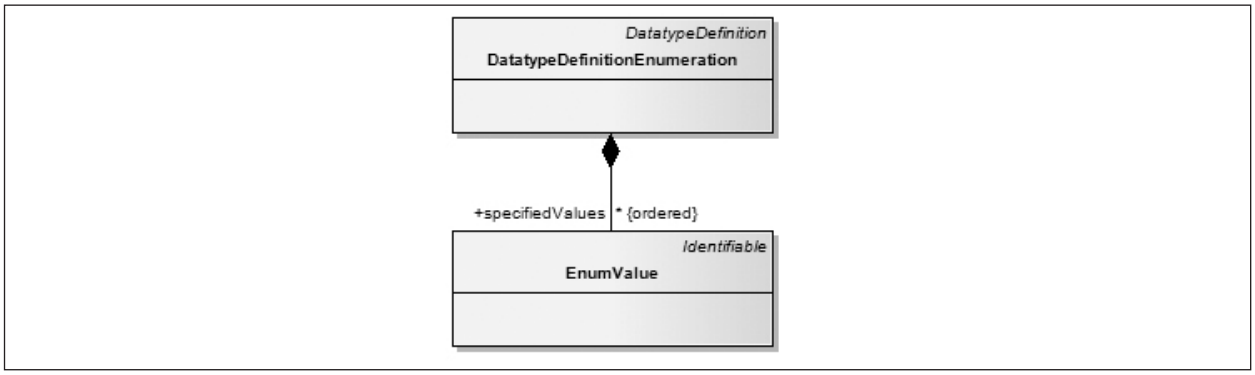


Figure 32: EnumValue nested inside DatatypeDefinitionEnumeration

Here, instances of the information type EnumValue become nested inside DatatypeDefinitionEnumeration (see Figure 32):

```
<DATATYPE-DEFINITION-ENUMERATION>
        …
        <SPECIFIED-VALUES>
            <ENUM-VALUE>
                …
            </ENUM-VALUE>
            <ENUM-VALUE>
                …
            </ENUM-VALUE>
            <ENUM-VALUE>
                …
            </ENUM-VALUE>
            …
          </SPECIFIED-VALUES>
        …
</DATATYPE-DEFINITION-ENUMERATION>
```

In the example above, the order of the aggregated information types is relevant. This is indicated by an additional {ordered} after the specification of the multiplicity of the aggregation.

NOTE: Composite subelements can be ordered, i.e. the subelements must appear in the sequence order defined by the value of the UML class tag order. For example, the order of the subelement of RIF is defined in Figure 33

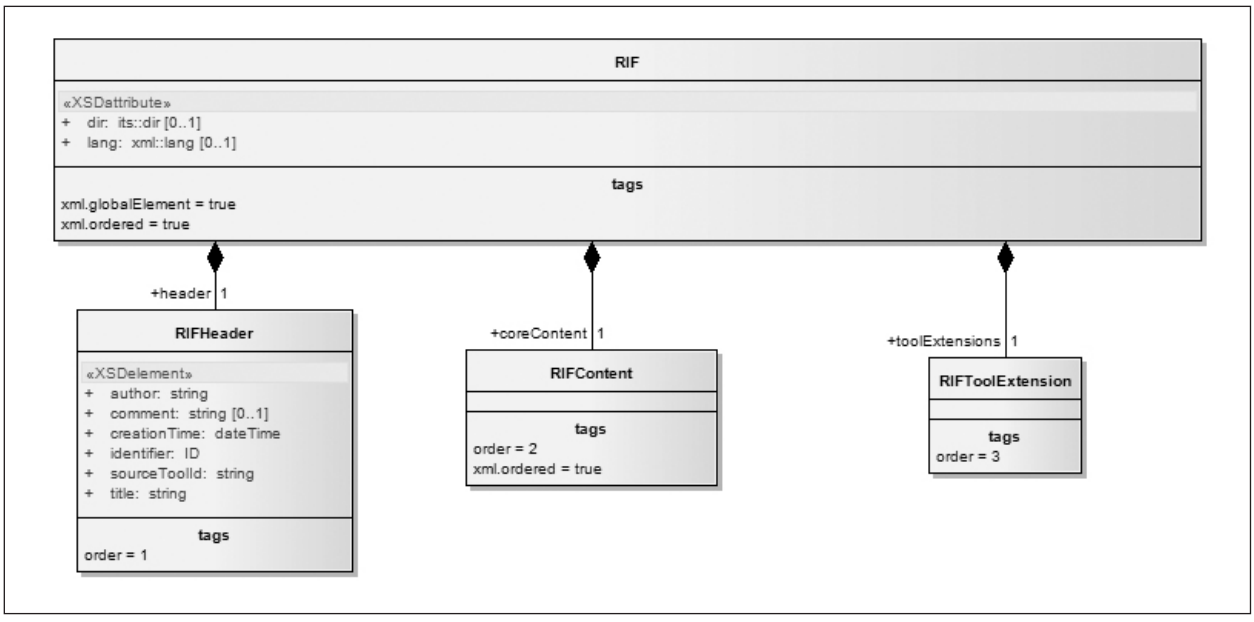


Figure 33: Order of RIF subelements

This results in this XML sequence:

```
<RIF>
        <HEADER>
                <RIF-HEADER>
                        …
                </RIF-HEADER>
          </HEADER>
        <CORE-CONTENT>
                <RIF-CONTENT>
                        …
                </RIF-CONTENT>
          </CORE-CONTENT>
        <!-- optional -->
        <TOOL-EXTENSIONS>
                <RIF-TOOL-EXTENSION>
                        …
                </RIF-TOOL-EXTENSION>
          </TOOL-EXTENSIONS>
</RIF>
```

The order of the subelements of RIFContent is defined in Figure 34.

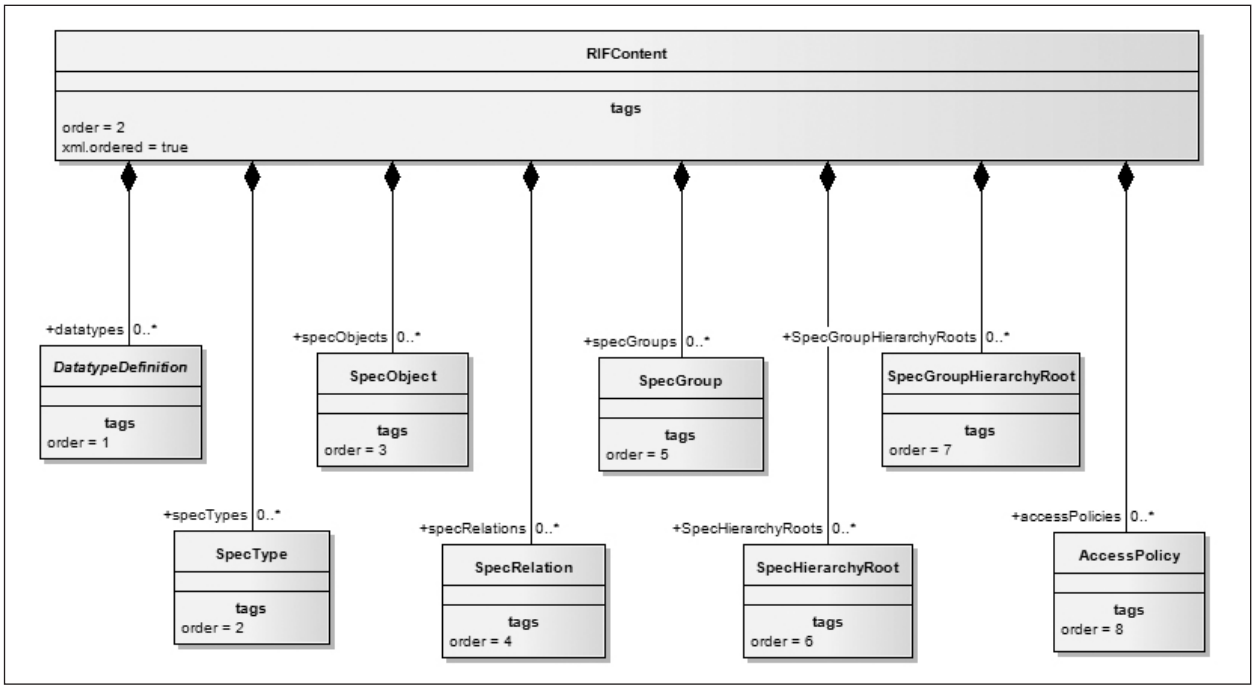


Figure 34: Order of RIFContent subelements

4.1.4 Association

The association mechanism is based on the unique identifier which is defined in the abstract superclass Identifiable. All information types that have to be referable inherit from Identifiable. The unique identifier is generated by the RM tool that is the origin of the related information element using an algorithm that assures global uniqueness (see also Section 3.2.3.1).

References are always wrapped by an XML element whose name is the role name of the aggregated information element(s) converted using the rule from Section 4.1.1.
A reference from an information element to another information element is made up of an XML element whose name is generated using the rule described in Section 4.1.1 and adding the postfix
"-REF". The unique identifier of the referenced information element is stored inside this XML element. The content of the XML element must conform to the W3C XML schema type xsd::IDREF.

Example:

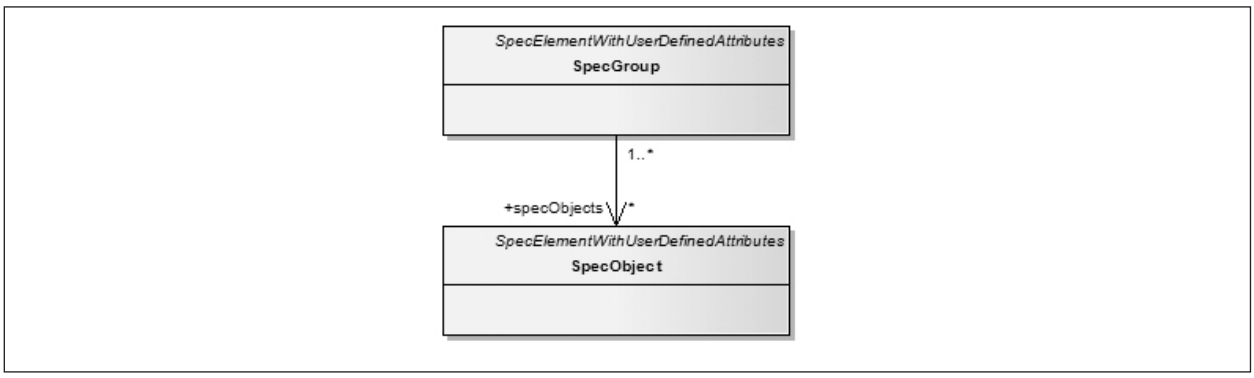


Figure 35: Example for an assosciation between two classes

Instances of the information type SpecObject are referenced by an instance of SpecGroup (see Figure 35):

```
<SPEC-GROUP>
        …
        <SPEC-OBJECTS>
                <SPEC-OBJECT-REF> … </SPEC-OBJECT-REF>
                <SPEC-OBJECT-REF> … </SPEC-OBJECT-REF>
                <SPEC-OBJECT-REF> … </SPEC-OBJECT-REF>
                …
        </SPEC-OBJECTS>
        …
</SPEC-GROUP>
```

4.1.5 Changing the XML namespace

The RIF exchange file contains XML elements whose content is not validated against the RIF-schema. These XML elements with contents that are validated against a different (e.g. the XHTML) schema are expressed in the RIF model in a special way.

Example:

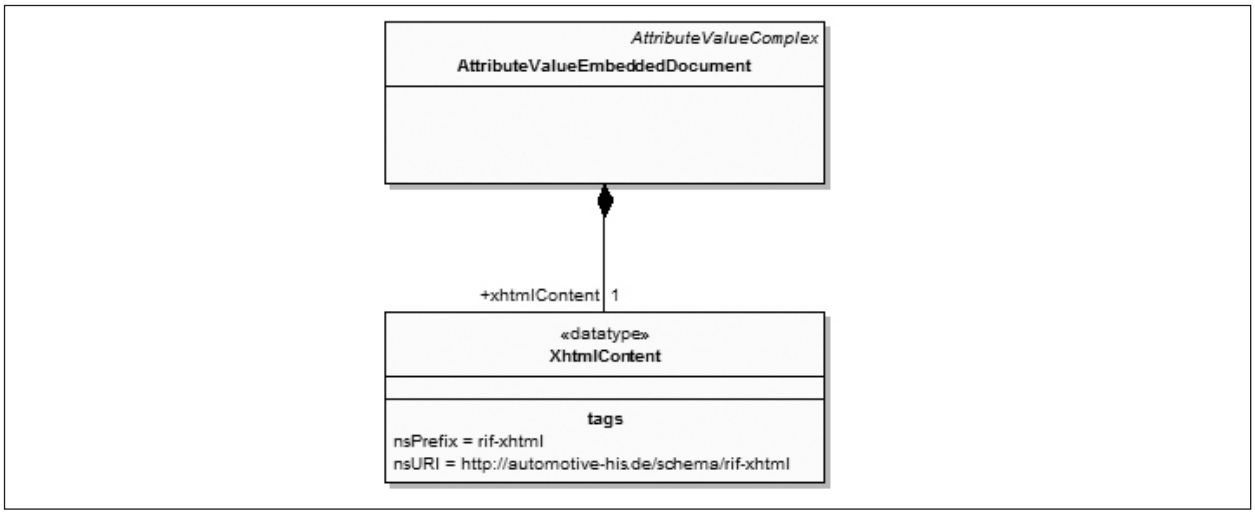


Figure 36: Example of how tagged-values are used to enrich the RIF metamodel with information

The ATTRIBUTE-VALUE-EMBEDDED-DOCUMENT element contains a sub-element XHTML-CONTENT with data in XHTML format. The XHTML-CONTENT element needs to be validated against the RIF-XHTML schema and is thus out-of-scope of the RIF-model. This is indicated in the RIF-model by an information type (here: XhtmlContent) with an additional stereotype <<datatype>> and an additional element tag nsURI containing as value the name space URI that is valid for the content. The name space prefix that must be used for the new name space is indicated by the value of the additional tag nsPrefix (shown in Figure 36).

4.2 XML Internationalization

4.2.1 Encoding of special characters

The lexical contents of objects in RM tools often include non-alphanumerical characters. Examples reach from currency symbols to scientific notations. Apart from that, the characters of certain languages contain special characters like umlauts in German or accents in French.

There is an established way to encode special character in XML that should be used in RIF XML documents as well: the referencing of Unicode characters.

Unicode characters can be included in any XML document by the use of character references.

For details, see the W3C recommendation

http://www.w3.org/TR/2004/REC-xml-20040204/#dt-charref

For the use of special characters of certain languages (for example: umlauts, accents and so forth), you may set the encoding information of the XML declaration of the RIF document.

Please note that parsers are only required to support UTF-8 and UTF-16 encodings.

Still, the ISO 8859 encodings are supported by many XML parsers, too.

4.3 Embedded objects

Embedded objects can be used within specification objects and denote those data objects that are not created by nor edited with the RM tool (e.g. Visio drawing, Excel sheet). These embedded objects can be inserted into a specification object by e.g. Copy-and-Paste via the clipboard or by Drag-and-Drop from another application.

Two groups of embedded objects can be distinguished:

- Simple embedded objects:
  These objects can be easily displayed or played-back within the RM tool (e.g. images, sound). Functionality for displaying or playing-back simple embedded objects can be easily added to RM tools.

- Complex embedded objects or OLE objects:
  OLE objects require an external program that can handle the specific data type in order to render and possibly edit the OLE object's content. OLE objects can usually be edited in the context of the embedding application (i.e. within the application window of the RM tool).

The RIF exchange file supports both simple and complex embedded objects which can be stored either within the RIF exchange file or in an external object file. A packaging functionality compresses and packs the exchange file together with the various external object files to provide easy handling.

It is recommended that storing embedded objects in an external file is the regular case. Storing embedded objects within the RIF exchange file rapidly blows up the size of the exchange file resulting in a significant decrease of the XML parsing speed. Thus, embedding objects within the exchange file should only be used if the data size of the object is very small.

4.3.1 Multiple representations of embedded objects

Certain OLE objects may not be displayable on any importing RM tool because the application associated with the type of the OLE object is not installed on the respective PC. For these cases, at least a simpler representation form of the OLE object has to be displayed as a substitution, e.g. a bitmap image as substitution for a vector drawing object.

RIF supports this feature by allowing multiple references from an embedded (OLE) object to the object data as shown in Figure 37.

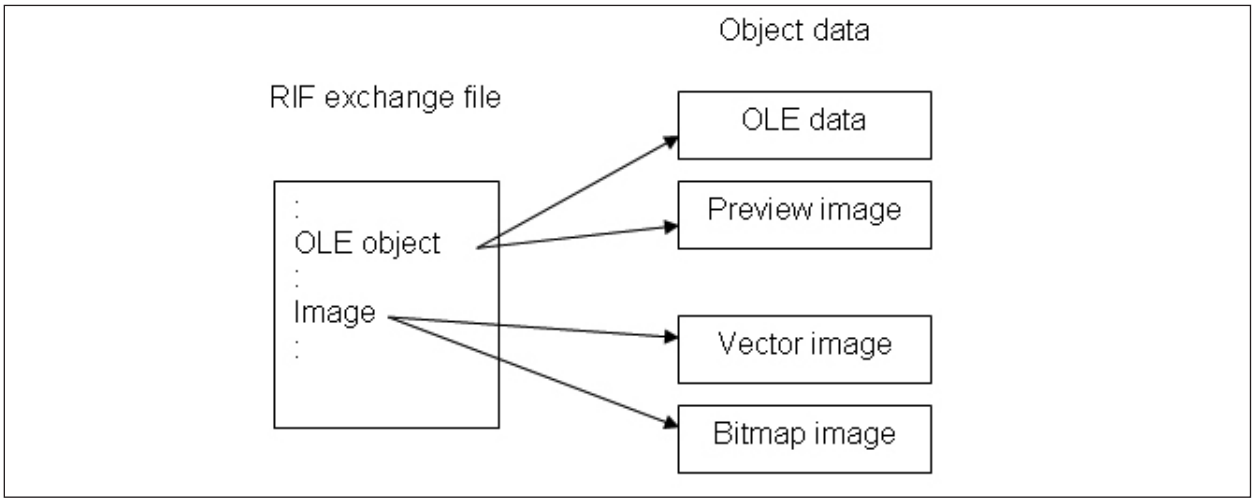


Figure 37: Multiple references of OLE objects

However, the RM tool is required to create and store the different representation forms of the embedded object in order for this feature to work.

4.3.2 Embedded objects in RIF

An embedded object can show up in the context of a specification text within a specification object. Specification text is stored within the XHTML namespace to keep text structure and text styles. Objects are embedded within this context using the XHTML object specification.

The XTHML object specification defines the Object element and additional attributes. For RIF, only a subset of these attributes is relevant and used. These attributes are shown together with their purposes in the following Table 5:

XML attribute	Purpose
Name	Unique identifier of the embedded object
Data	Reference to external object data file as URI (not used if object data is stored in the RIF exchange file)
Type	Data format (for simple embedded objects)
classid	CLSID (for OLE objects)
height	Height in pixels if object is an image (optionally)
Width	Width in pixels if object is an image (optionally)

Table 5: RIF-XHTML object attributes and their purposes

Multiple representation formats for the same embedded object are realized by multiple, contiguously located Object elements that have the same identifier (name attribute) but refer to different object formats and data. The RM tool is responsible to create the identifier of an embedded object that must be unique at least within the RIF exchange file. The order of the references to the multiple representation forms must be of decreasing completeness in terms of information content, i.e. the most complete representation form must occur first.

If the object data is to be stored within the RIF exchange file, then the data is encoded in the Base64 format and placed inside the Object element as content of the element.

Examples:

<RIF-XHTML:OBJECT name="**identifier1**" classid="**00021A13-0000-0000-C000-000000000046**" data="**My_drawing.vsd**"/>

<RIF-XHTML:OBJECT name="**identifier1**" type="**CF_BITMAP**" data="**My_drawing.gif**"/>

<RIF-XHTML:OBJECT name="**identifier2**" type="**CF_BITMAP**" data="**Very_small_picture.gif**">
d6ZkC82F0mP… […Base64 encoded data…]
</RIF-XHTML:OBJECT>

4.4 Packaging

If many OLE objects are used in a specification document, the RIF export results in one exchange file and many files containing the embedded object data. Handling of these files would be laborious if they were not packaged into one file. Thus, RIF specifies packaging of the RIF exchange file together with the embedded object files to provide for easy handling and to reduce the size of the resulting package file.

RIF packaging results in a RIF exchange package which is based on the ZIP file format as specified in http://www.pkware.com/appnote.html.

Within the RIF exchange package, the RIF exchange file must be named rif_ef.xml. The embedded object files can have arbitrary names with the only limitation that these names must be valid file names on Microsoft Windows operating systems.

The packaging mechanism should not be used to transport the structural information like RM tool packages. Instead, the SpecGroupHierarchy mechanism should be used.

4.5 Tool Interfaces

A RIF export for a RM tool should allow the user to specify which requirements should be allowed. Some RM tools provide the concept of views or filters on the managed requirements. All requirements viewed in such a view may be used by a RIF exporter to select the set of requirements to export.

Some RM tools support the notion of deleted requirememts, but this is explicitly not supported by RIF. But a RIF exporter or tool may use a compare algorithm to determine the deleted requirememts between different versions of a RIF document

4.5.1 Guide to implement a user interface

This is a proposal to guide the user through a workflow of dialogs and information how to set parameters for the transfer process.

A transfer process is divided into export and import whereas both sub-processes must be repeatable in several ways. An export is repeatable as a re-export of the same set of requirement elements and attribute subsets as a new version containing the current modifications. A re-export of old versions is not inherently supported by RIF. Exporting old versions must be covered by the tool feature for restoring older versions from the history data.

An import process involves a creation and a restore or update transaction to generate a (new) version of requirements data within the target tool.

To manage this repeatability it is necessary for both tools participating in the exchange to save additional configuration settings for each transfer process.

4.5.1.1 Sample export process

1. A project member (sender) runs an Export Wizard
2. The sender selects a set of requirement documents containing the requirements and the subset of attributes to interchange. The sender sets access policies and access rights to allow the recipient to edit certain attributes In case of a re-export the sender can load an existing configuration of all settings from file or database and continue with step 6 if desired
3. The sender sets RIF-file based information e.g. comments, document title, validation date
4. In extension, the sender sets re-usable project or organisational meta information stored in the tool to structure and organize different sets of interchange configurations
5. The sender sets destination path and RIF package name
6. The tool generates its source Tool-ID to support enhanced importing features on the recipient's tool
7. The tool generates SO Interchange-Ids for all data elements for a first interchange or re-uses existing SO Interchange-ID's in case of a re-export
8. Save settings including SO Interchange-Ids in the tool database or a file at a user-defined location to be able to re-export the same set of requirement data later
9. Start export and save requirement data into the RIF package

4.5.1.2 Sample import process

1. A project member (recipient) runs an Import Wizard
2. Select a name and path to the RIF package to be imported
3. The tool verifies consistency of the RIF exchange file according to the XML schema
4. The importing tool checks the source tool-ID to switch on enhanced importing features it may support
5. For the first import: A decision how and where to integrate the RIF exchange file content within the target tool element structure is made by the user
6. In case of re-import: The tool compares the scope of the RIF exchange file with the one defined in the stored setting file to identify new or deleted entries.
7. In case of re-import: The tool evaluates the changes between the versions of the requirement content in the RIF exchange file and the internal database. It indicates violations of the access rights defined in the setting file which was used for the first export/import. Optional, changes made by the sender are accepted or rejected with a new version
8. Execute import (errors are handled)
9. Store the scope and settings of the imported RIF-package in the tool database or a file at a user-defined location to re-use it for a later re-export.
10. Create a log file with all settings, warning, and error messages

4.5.1.3 Warnings and error handling

A RIF interface implemented in an RM tool should give warnings when errors occur. In the following, some obvious error cases are given as examples:

Special export warnings and errors:
– Versioning: Re-export of an already old version
– Modified export configurations
– Empty attributes must be provided with default values

Special import warnings and errors:
– Versioning: Re-import an already integrated (old) version
– Logical content:
  o Changes made to the exported set of access policies
  o Missing default values for all transferred elements and attributes
  o Deleted requirements since previous transfer process

Appendix A: Additional Resources

This appendix contains a listing of additional resources that are relevant for the implementation of RIF.

4.6 RIF XML Schema (rif.xsd)

A current version of the RIF XML Schema can be downloaded from the HIS-Homepage http://www.automotive-his.de/rif/doku.php

4.7 RIF XHTML Schema

A current version of the RIF-XHTML Schema can be downloaded from the HIS-Homepage
http://www.automotive-his.de/rif/doku.php

4.8 RIF Test Data

Sample RIF XML files are available and can be downloaded from the HIS-Homepage
http://www.automotive-his.de/rif/doku.php

Appendix B: Glossary

English Terms and Definitions	Meaning / Explanation	Synonym / Homonym
List of Abbreviations		
AD	Attribute Definition	-
AO	Attribute Object (outdated)	Homonym: - AV.
AP	Access Policy	-
AS	Answer Specification	-
AT	Attribute Type	-
AV	Attribute Value	-
AccPol	Access Policy	-
AnSpec	Answer Specification	-
AttDef	Attribute Definition	-
AttVal	Attribute Value	-
CRS	Customer Requirements Specification	-
DB	Database	-
DT	Data Type	-
DTD	Document Type Definition	-
EF	Exchange File	-
EFF	Exchange File Format	-
EP	Exchange-Process	-
ID	Identity	-
InfoElem	Information Element	-
InfoType	Information Type	-
InitSpec	Initial Specification	-
NFR	Non-Functional Requirement	-
OrigSpec	Original Specification	-
OEM	Original Equipment Manufacturer	-
PP	Project Partner	-
PreSpec	Previous Specification	-
RD	Requirement Data	-
RO	Relation Object (outdated)	Homonym: - SpecRel.
RT	Requirement Type	-

English Terms and Definitions	Meaning / Explanation	Synonym / Homonym
RelGroup	Relation Group	-
RelType	Relation Type	-
Req	Requirement	-
ReqSpec	Requirement Specification	-
ReqSpecDoc	Requirement Specification Document	-
RIF	Requirements interchange Format	-
SA	Specification Attribute	-
SE_A	Specification Element with User Attributes	-
SG	Specification Group	-
SH	Specification Hierarchy	-
SHO	Specification Hierarchy Object	-
SHR	Specification Hierarchy Root	-
SO	Specification Object	-
SOIID	Specification Object Interchange-ID (SO-Interchange-ID)	-
SR	Specification Relation	-
ST	Specification Type	-
SpecElem_A	Specification Element with User Attributes	-
SpecGroup	Specification Group	-
SpecHierarchy	Specification Hierarchy	-
SpecHierarchyRoot	Specification Hierarchy Root	-
SpecGroupHierarchy	Specification Group Hierarchy	-
SpecGroupHierarchyRoot	Specification Group Hierarchy Root	-
SpecObject	Specification Object	-
SpecRelation	Specification Relation	-
SpecType	Specification Type	-
SuD	System under Development	-
SysRS	System Requirements Specification	-
ToC	Table of Content	-
UpSpec	Updated Specification	-
WP	Work Package	-

English Terms and Definitions	Meaning / Explanation	Synonym / Homonym
Object-Words		
Access Policy	The AccPol defines Access Rights. For this reason, AccPol is related to 10 classes of the EFF: 1. SpecHierarchy, 2. SpecHierarchyRoot, 3. SpecObject, 4. SpecRelation, 5. SpecGroup, 6. SpecType, 7. Relation Group, 8. DatatypeDefinition, 9. Attribute Definition, 10. Attribute Value. AccPol is an optional part of the EFF, while the default will be "read only".	see also: - Access Right, - Modification Right, - SH, - SHR, - SO, - SR, - SG, - ST, - RelGroup, - AD, - XML Schema, - AV, - EFF. Synonym: - Authorisation.
Attribute	An Attribute is assigned to an Object (for instance an SO a Group) and enlarges the information that is available to its assigned Object.	-
Attribute Definition	An AttDef. is valid for a (group of) SE_A. For this (group of) SE_A(s), an Attribute Definition defines 2 things: 1. The name of the Attribute it defines, and 2. The Attribute Type. Any AttDef. encloses a default value for any AT. Each AD is aggregated by exactly one SpecType. Each AD is associated with exactly one XML-Schema (here: AT). Each AD aggregates one or zero (0..1) Attribute Value. Each AD is associated by exactly one Attribute Value.	see also: - SE_A, - AT, - ST, - AV.
Attribute Type	Each AV is an instance of an AT.	see also: - AO, - DT. Dissociate from: - AttDef.
Attribute Value	Information typed by an AT. An AV can give information of a value or its reference if the AV is used for expressing Relations.	see also: - AT, - AO, - Attribute. Homonym: - Attribute Object (outdated).

English Terms and Definitions	Meaning / Explanation	Synonym / Homonym
Character Set	The EFF supports the definition of Character Set(s) to be used for displaying the Requirements included in an EF like UTF8, UTF16, ISO8859, ANSII, Unicode for instance.	see also: - EFF, - EF. Homonym: - CharSet.
Class	See "Requirement Type".	Homonym: - Requirement Type.
Class "Identifiable"	A special Specification Type with the purpose to identify all Specification Objects that are suitable for the transformation process. Note: Some of its Attributes are "LongName" and "Identifier.	-
Country Code	The EFF shall enable to add a Country Code to any EF, for instance for the sake of displaying numeric values or dates in an unambiguous, familiar way.	-
Data Type	The Data Type of each Specification Attribute defines the values a Spec Attribute might have.	Synonym: - Type.
Exchange File	Means of transferring (parts of the) Requirement Specification Document(s) via an Export- and Import-Process between the RM-Tools of the Customer(s) and the Supplier(s) to exchange Requirement Specifications, while each EF can contain information of multiple origins. Each EF meets the Requirements of the EFF.	see also: - Exchange File Format, - SHR.
Exchange File Format	The format of the Exchange File(s). The EF can be regarded as an instantiation of the EFF. The EFF specifies the data structure for storing requirements together with e.g. comments, answers, links and history. While the format extensively specifies all necessary data structures, an EF according to the EFF may contain only a part of the structures defined in the EFF. Actually it is intended that the EFF shall support accessibility/ visibility, ownership and changeability information, which includes the necessity to support user-roles as well. The EFF supports Country Codes as well. The EFF supports the definition of Character Set(s). The EFF supports means to avoid any data contradiction. The EFF must be easily convertible to other appropriate formats and structures, for instance using XSLT. The method how the EFF will be developed must derive from the AutoSAR method.	see also: - Exchange File, - exchange, - Country Code - Character Set
Export-Process	A process to extract Requirement Specification Document(s) from the RM-Tool to a file in EFF.	see also: - Import-Process - Partially Export - Incrementally Export.

English Terms and Definitions	Meaning / Explanation	Synonym / Homonym
Export Interface	The Export Interface must support Export-Processes that create EF in EFF. Vendors of RM-Tools are invoked to implement a suitable interface that fulfils the RIF-Requirements.	-
GUID	See Specification Object Interchange-ID	Synonym: - GUID
Identity	Each ID is related to a Requirement (or any other Specification Object of an EF) and represents a means to identify the assigned Requirement in an unequivocal way. IDs must be available for Attributes and Types of Attributes.	-
Import-Process	A process to include Requirement Specification Document(s) into a RM-Tool's DB. Counterpart of the Export-Process. After importing a ReqSpec in EFF, the ReqSpec is in the DB of the importing party's RM-Tool and mapped to its content. Main goal of the import part of an Exchange-Process is to enable the importing contractual party to comment, question and answer Requirements and Comments of any other partner while traceability is ensured, simultaneously.	see also: - importing - Export-Process
Import Interface	The Import Interface must support Import-Processes that create EF in EFF. Vendors of RM-Tools are invoked to implement a suitable interface that fulfils the RIF-Requirements.	-
Information Element	Information Elements are all the Objects enclosed in an EF. Every InfoElem is an instance of a InfoType.	see also: - InfoType.
Information Type	Every Information Element is an instance of a Information Type.	see also: - InfoElem.
Inheritance	Classes inherit state and behavior from their superclasses. Inheritance provides a mechanism for organizing and structuring software programs.	-
Interface	Means of a RM-Tool to export and import EF.	see also: - Export Interface, - Import Interface.
Layout	A Layout defines, how the Requirements are arranged in any kind of coherent Document, for instance for a print-out. Layout information have their own Version.	see also: - Style.
Link	A means to explicitly communicate logical or structural dependencies among Requirements.	Homonym: - Relation.
Mapping-Process	Between the exporting and the importing RM-Tool a mapping for the individual attributes must be specified. The Mapping-Process is part of the Merging-Process and related to the technical aspect of merging data. To ensure this process runs automatically, a Mapping Table exists.	see also: - Mapping Table, - Exchange-Process, - Merging-Process.

English Terms and Definitions	Meaning / Explanation	Synonym / Homonym
Mapping Table	A Mapping Table defines for each Object within an EF where it must be stored in the DB of the importing RM-Tool and vice versa. This mapping must be configurable.	Synonym: - Mapping Tabelle.
Meta Information	Additional Information on Specification Objects needed for processing by the RM-Tools and/or the EF. This kind of information is invisible for the User by default.	-
Priority	How critical a Requirement is to the SuD.	-
Relation	A means to explicitly communicate logical or structural dependencies among Requirements. A RT defines for its instances, the RO's, which Attributes the Relation has.	Homonym: - Link. see also: - RT, - RO.
Relation Group	A RelGroup defines whether it is allowed to create and add new Relations to the Requirements in the corresponding RelGroup. A RelGroup restricts the creation of Traces starting at the parent elements of their SpecGroup they belong to.	see also: - SO, - SG, - SR, - ST.
Relation Type	Each Relation Object is assigned to a Relation Type. A Relation Type contains information about its Cardinality. Multiple Relation Types can be used for different kinds of Relations. Any RelType is realised by a SpecType.	see also: - SpecType.
Requirement	This term constitutes a fundamental, logical element of Requirements Engineering (see www.hood-group.com for details). Requirements are collected in Requirement Specifications.	see also: - Requirement Specification.
Requirement Data	The Specification(s) are organised in exchangeable RM-Tool data.	-
Requirement Type	Each SO is an instance of a RT. A multitude of RT shall be able to define for the EFF. Each RT contains 3 things: 1. it's Name, 2. its ID, 3. the defined Attributes per instantiated SO. Any ReqType is realised by a SpecType.	Synonym: - Class. see also: - SO, - SpecType.
Requirements Interchange Format	This term is a Homonym to EFF. While "EFF" referred to the technical aspect of name-giving to the SuD of this project, the term "RIF" satisfies, for instance, marketing aspects as well. "EFF" can be seen as a placeholder-term during the development process, while "RIF" must be seen as the final, consolidated term.	Homonym: - EFF. see also: - SuD.

English Terms and Definitions	Meaning / Explanation	Synonym / Homonym
RM-Tool	A SW-Tool for specifying and managing Requirements, explicitly with the possibility to exchange Requirements. A RM-Tools does not need to be available at all contracting parties, but the functionalities are mandatory. The RM-Tool is built upon and around a DB where all the Specifications and parts of Specifications are edited and finally integrated. This integration is done within the RM-Tool's DB on Customer's side.	-
Root Element	A Root Element is the structural and logical basis of every EF.	see also: - EF, - TimeStamp, - XML-Schema, - ST, - SO, - SR, - SG, - SHR.
Requirement Specification	A means to define the goals the SuD has to meet when the development is finished. This goals are defined by single Requirements, organised in a Requirements Specification. For this project, two basic types of Requirements Specifications exist: CRS and SysRS. A ReqSpec usually exists within the DB of RM-Tools, RSD's or printout versions of a RSD, while the latter two can be part of an Exchange-Process.	see also: - SuD, - CRS, - SysRS, - RSD, - Exchange-Process. Synonym: - Specification, - Spec.
Requirement Specification Document	A Document containing Requirement Specifications. Each ReqSpecDoc can exist in two ways: a) as a EF, b) as a printed EF and can be part of an Exchange-Process. Within such a process, a RSD can be of one of the following Specification types: - Initial Specification, - (Original Specification), - (Previous Specification), - Answer Specification, - (Updated Specification), while the terms in brackets refer to Synonyms.	see also: - SysRS, - CRS, - RS, - EF, - Exchange-Process.
Requirements Group	The EFF supports clustering of Requirements in Groups. Each Group has its own Version-Information.	Synonym Specification Group, - SG.
SourceTool ID	ID to identify the RM-Tool used, defined in the "RIF" root class of the RIF-model.	

English Terms and Definitions	Meaning / Explanation	Synonym / Homonym
Specification Attribute	A Specification Object has 1 to n Attributes (one for the ID is mandatory). Specification Attributes are based upon the related Specification Type of the Specification Object it belongs to. User defined Attributes exist as well as Interface-relevant Attributes.	Spec Attribute, SpecAttribute.
Specification Element with User Attributes	The SE_A is a primary object within the EFF. It inherits its (user defined) Attributes to 4 Classes: 1. SpecObject, 2. SpecRelation, 3. SpecGroup, 4. SpecHierarchyRoot.	see also: - EFF, - SO, - SR, - SG, - SHR, - ST, - AV.
Specification Group	A SpecGroup is a means to combine SpecObjects in groups. Each Spec Object must be member in a minimum of one (1..n) SpecGroup(s). Each SG can associate 0 to m (0..m) SO(s). Each SG aggregates 0 to x (0..x) RelationGroups, while each RelGroup is aggregated by exactly one SG. A SpecGroup inherits from SE_A. Each SO has its own Version-Information. Each SpecGroup inherits to Attributes of SpecGroups. Each SpecGroup inherits to Attributes of RelGroups. A SpecGroup does not inherit content of Requirements and its Attributes.	see also: - SO, - SE_A.
Specification Hierarchy	Each SpecHierarchy represents exactly one sub-headline of the ToC of the EF where it is included. A SpecHierarchy contains information, related to the structure of the EF, not related to the content of the EF. It contains information whether the related chapter of the EF is allowed to supplement and/or alter.	Synonym: - Hierarchy, - Hierarchie, - Spec Hierarchy, - SpecHierarchy, - Tree-Structur, see also: - EF, - SHR, - SH.
Specification Hierarchy Object	One element of a Specification Hierarchy. Each Spec Hierarchy Object associates exactly 1 Spec Object.	Spec Hierarchy Object.
Specification Hierarchy Root	Each EF contains exactly one SHR. A SHR represents the chapter structure of the EF it is enclosed within. Each SpecHierarchyRoot defines whether main-headlines are alterable and whether the name of the hierarchy is alterable. SpecHierarchyRoot inherits to main-headlines and to the Attributes of the Hierarchy. A SpecHierarchyRoot has no impact on the content (Requirements and Attributes) of the EF and on the Attributes of a SpecGroup.	see also: - EF, - EFF, - AP, - SG, - SH.

English Terms and Definitions	Meaning / Explanation	Synonym / Homonym
Specification Group Hierarchy	Each Spec Group Hierarchy respresents a structuring information about Spec Groups.	
Specification Group Hierarchy Object	One element of a Specification Group Hierarchy. Each Spec Group Hierarchy Object associates exactly 1 Spec Group.	Spec Group Hierarchy Object.
Specification Group Hierarchy Root	Each Spec Group Hierarchy Root respresents the top level structuring of Spec Group Hierarchies.	
Specification Objekt	Atomic Object of the Exchange Format (and the Source- and Destination-RM-Tool as well due to Exchange-Process!) that is to be referenced by the RM-Tool in question. Each SO owns 5 basic information of: 1. its content, 2. its datatype, 3. its structural dependencies, 4. (1..n) related SA's. 5. its Version/History/Time-Stamp. To give an example, a SO can consist of text, picture or an OLE-Object. Every SO is instance of a ST, while the ST determines, which Attributes each SO owns. Every SO inherits to Attribute Values.	Synonym: - Requirements Object. see also: - AV, - SO, - SG, - SH, - ST. Dissociate from: - SpecHirearchyRoot.
Specification Object Interchange-ID	Special form of ID. For several exchange transactions reusable unique identity. It is created once by the export interface and unique for all following transactions of the same Requirements Specification Document and its Specification Objects and Attributes.	Synonym: - GUID
Specification Relation	Every Relation between ROs is represented by SpecRelation(s). Each SpecRelation is member in a *minimum of one* RelationGroup(s). Each SpecRelation associates *exactly two* SO, while one of the SRs has the role of the *source* of the relation, and the other has the role of the *target* of the referred relation. Every SR inherits to Attribute Values. Every SpecRelation inherits from SE_A.	see also: - RO, - SO, - AV, - SE_A.
Specification Structure	The structure of the Specification in question is to be generated in the RM-Tool of the Customer and can be split-up and send due to several exporting on Supplier's side. The Customer(s) generate(s) a Structure for the Specification and sends it to the Specification Writer(s).	SpecStructure, Spec.Structure.

English Terms and Definitions	Meaning / Explanation	Synonym / Homonym
Specification Type	The SpecType defines its own name and is the data basis for any SpecObject. A SpecType inherits to AV(s) and is allowed to add and delete Attribute Definitions. The ST defines the Attributes a SO is allowed to have. Each SO is one instance of a ST. A ST aggregates 0 to n (0..n) Attribute Definition(s). A ST is associated by 0 to m (0..m) Spec Element With User Attribute(s). SpecTypes realise *RelTypes* and *ReqTypes*.	Synonym: - Classes. See also: - SO, - AD, - SE_A, - RelType, - ReqType.
Style	A Style defines the appearance of characters, for instance bold, italic, underline.	See also: - Layout
System Requirements Specification	It describles the whole SuD in a more technical oriented language. It is devided into Sub System Requirements like sub-assembly or component specification.	-
System under Development	general: The (HW- and/or SW-) System or SubSystem the Customer wants to develop or define requirements for. The SuD is defined in the Contract between Customer(s) and Supplier(s). here: The SuD is the EFF, respectively RIF.	-
Table	An ordered columnar display of data.	-
Table of Content	Like in most books, each EF has its own ToC, represented by an SHR. A SHR aggregates 0 to n (0..n) SH(s).	see also: - EF, - SHR, - SH.
Time-Stamp	Every SpecObject has its own TimeStamp, showing the last point-in-time when it has been altered. defined in the "RIF" root class of the RIF-model: 1. date of creation, 2. time of creation, defined in the "Identifiable" class of the RIF-model: 3. date of last modification, 4. time of last modification,	see also: - Version, - History. Synonym: - TimeStamp.
Type-Definition	Type-Definitions exist for Requirements and Attributes. These Type-Definitions must be part of the EFF. During the Export-Process, the Types must be reducible to individually defined Subsets. Transferred Type-Definitions have their own version. During the Import-Process, the importing Interface must verify whether the value of its included Objects meet their Type-Definition. Additionally, the origin of any Type-Definition must be enclosed in the EFF. Due to the fact that Type-Definitions are supposed to be alterable, those changes must be traceable throughout an EF.	-

English Terms and Definitions	Meaning / Explanation	Synonym / Homonym
Version	Each Spec Object, Spec Group and Spec Hierarchy has its own Version, coded as a timestamp according to ISO 8601.	Differentiation to Timestamp and History!
View	A view is a particular way of looking at a set of data. Typically, a view arranges the records in some order and makes only certain fields visible	-
Work Package	A defined task to be fully processed by a defined person until the WP's Due Date.	see also: - Due Date
XML DocBook Format	A special XML dialect for book	-
XML-Schema	An XML-Schema provides a means for defining the structure, content and semantics of XML documents. XML-Schema is used as one representation of the EFF. It can be used to check the validity of EF(s).	Homonym: - Schema.
XSLT	*Extensible Style Language Transformation*, the language used in XSL style sheets to transform XML documents into other XML documents or dialects.	-
Roles / Actors		
Customer	The Customer(s) is placing the order to develop the SuD by the Supplier(s). The Customer instantiates the Contract. The Customer(s) is the over-all owner of the Specification(s). The Customer(s) uses a RM-Tool. The final version of the Requirements are to be found within the Customers' RM-Tool's DB.	Synonym: - OEM, - Manufacturer.
Lector	Proof-Reader, engaged by the Customer(s). A Proof-Reader is part of the PP.	Homonym: - Proof-Reader.
OEM	Original Equipment Manufacturer	-
Project Partner	Each of the Customer(s) and of the Supplier(s) listed in the Contract related to build the SuD. Within this Contract, the role of every Customer and Supplier in the development process of the SuD is defined. To divide several PP's, a number might be added, for example PP1 and PP2 might be partners involved in an Exchange Process.	see also: - Customer - Supplier - Contract
Primary Actor	The Trigger for a UseCase. The Primary Actor is one defined party of the Customers involved.	see also: - Trigger
Proof-Reader	A Proof-Reader is part of the PP.	Homonym: - Lector. see also: - PP.
Review	An inspection or examination for the purpose of evaluation or improvement.	see also: - Reviewer

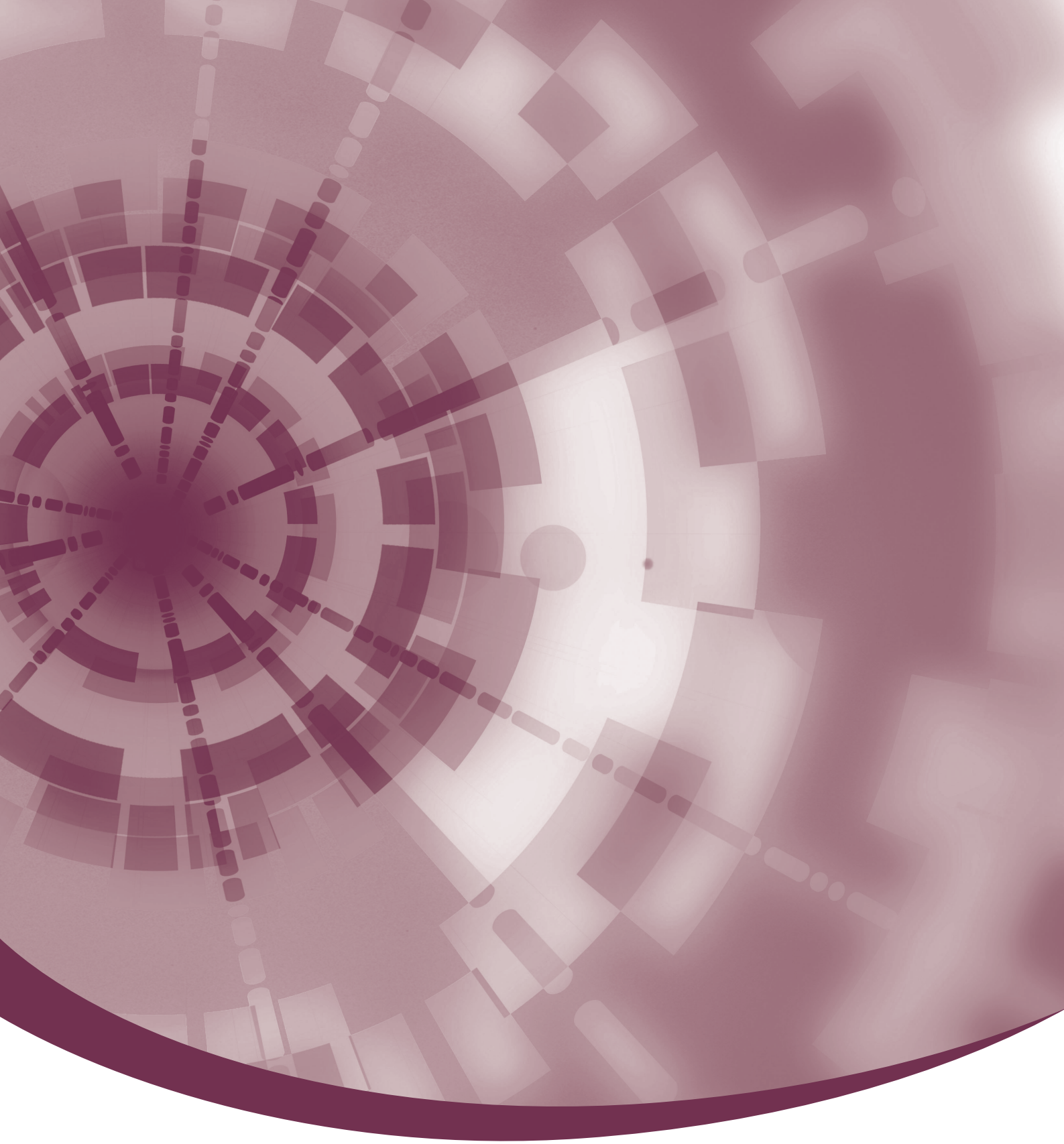
English Terms and Definitions	Meaning / Explanation	Synonym / Homonym
Reviewer	One or more people who are involved in the development process of the SuD for the sake of ensuring quality. A Reviewer might be internal (i.e. Customer) or external (i.e. Supplier, Proof-Reader), but must be part of the group of PP.	see also: - Review - reviewing
Specification Writer	Any person on Supplier(s)'s and/or Customer(s)'s side who is authorized to write (parts of) Requirement Specification Documents and is authorised to exchange them with the Customer. Only the Specification Writer(s) on side of the Customer needs a RM-Tool installed.	-
Sub-Contractor	A contractual party, involved in the development process for the SuD.	-
Supplier	One or more company(s) that is involved by the Customer(s) in a project in order to develop or specify parts of the SuD.	Specification Writer, Sub-Contractor.
User	A person or a pice of SW that has access to the RM-Tool and operates it.	-
Tools		
Borland - CaliberRM™	http://www.borland.com/caliber/	
IBM - RequisitePro™	http://www-306.ibm.com/software/de/rational/	
Microsoft® Word	http://www.microsoft.com/germany/default.aspx	
TCP IA - IRqA™	http://www.irqaonline.com/	
Telelogic - DOORS™	http://www.telelogic.com/	
MKS - Integrity™	http://www.mks.com/	
Trademarks		
DOORS/ERS™	DOORS/ERS™ is a trademark of Telelogic AB	
IRqA™	IRqA™ is a trademark of TCP Sistemas e Ingeniería S.L. Spain	
CaliberRM™	CaliberRM™ is a trademark of Borland Software Corporation	
Microsoft® Word™	Microsoft® Word™is a trademark of Microsoft Corporation	
RequisitePro™	RequisitePro™ is a trademark of IBM Corporation	

Table 6: List of abbreviations, object-words, trademarks and tools

Appendix B: Glossary

Document version	Modification	Date
1.0	Initial version	March 2005
1.0a	• Added "identifier" attribute in root information type • Removed "checksum" attribute from root information type since checksum functionality is used from ZIP packaging • "SOURCE" and "TARGET" references of type SpecRelation changed to lower-case "source" and "target" • Updated description and class diagrams for complex datatype to reflect bug fix • Updated description for content wrappers	November 2005
1.1	• Updated RIF-XHTML description • Corrected notation of elements (concrete and abstract) • Corrected various other little flaws	October 2006
1.2	• New RIFHeader, RIFContent and RIFToolextension Element • xml:boolean and all other XML standardized fundamental data types instead of Boolean and the other RIF fundamental data types • new xml:lang element in RIF element • new its:dir element in RIF element • remove RIF::CountryCode element • Order of RIF top level elements • DataTypeDefinitionDate: custom date time format is dropped • new SpecGroupHierarchy and SpecGroupHierarchyRoot element • identifier type changed to xsd::ID und xsd::IDREF	July 2008

Table 7: RIF modelling history



ProSTEP iViP Association
Dolivostraße 11
64293 Darmstadt
Germany

Tel. +49-6151-9287336
Fax +49-6151-9287326

psev@prostep.com
www.prostep.org

ISBN 978-3-9811864-8-2

Version 1.2, October 2008