

RESEARCH REPORT

VTT-R-04722-15



Implementation of ReqIF for the Simantics platform

Authors: Teemu Mätäsniemi, Jarmo Alanen

Confidentiality: Public

Report's title		
Implementation of ReqIF for the Simantics platform		
Customer, contact person, address		Order reference
Tekes, Matti Säynätjoki Kyllikinportti 2, P.O. Box 69, FI-00101 Helsinki, FINLAND		Tekes: 40204/12
Project name		Project number/Short name
Computational methods in mechanical engineering product development		100553 (78634)/SIMPRO
Author(s)		Pages
Teemu Mätäsniemi, Jarmo Alanen		22/-
Keywords		Report identification code
Requirements engineering, Requirements management, ReqIF, Simantics, XML, XML Schema		VTT-R-04722-15
Summary		
<p>This report is the result of task 3 ('Requirement- and simulation driven design') of the SIMPRO project ('Computational methods in mechanical engineering product development') carried out during the years 2012–2015. The report focuses on the steps to support ReqIF data roundtrip and data validation in the Simantics environment. ReqIF and an example use case to exchange ReqIF data between two companies are introduced. In addition, applicability of different XML schema languages for data validation in the Simantics environment is considered. Possible approaches and technologies to resolve ReqIF data (XML data) exchange are presented. Finally, implemented features in the Simantics environment are demonstrated.</p>		
Confidentiality	Public	
Tampere 20.10.2015		
Written by	Reviewed by	Accepted by
Teemu Mätäsniemi, Research Scientist	Marko Luukkainen, Research Scientist	Riikka Virkkunen, Head of Research Area
VTT's contact address		
VTT Technical Research Centre of Finland, P.O. Box 1300, FI-33101 Tampere, Finland		
Distribution (customer and VTT)		
Tekes/Matti Säynätjoki, 1 original VTT/archive, 1 original		
<p><i>The use of the name of VTT Technical Research Centre of Finland Ltd in advertising or publishing of a part of this report is only permissible with written authorisation from VTT Technical Research Centre of Finland Ltd.</i></p>		

Preface

This report is the main result of VTT subproject Task 3 ('Requirement- and simulation driven design') of the SIMPRO project ('Computational methods in mechanical engineering product development') carried out during the years 2012–2015 and financed by the following organisations: Tekes (the Finnish Funding Agency for Innovation, Aalto University, Tampere University Technology, Lappeenranta University of Technology, University of Jyväskylä, Wärtsilä Finland Oy, Patria Land Systems Oy, KONE Oyj, MeVEA Oy, FS Dynamics Finland Oy Ab, EDR & Medeso Oy, Dassault Systèmes Oy, Techila Technologies Oy and VTT Technical Research Centre of Finland Ltd. The project was coordinated by VTT.

Tampere 20.10.2015

Authors

Contents

Preface.....	2
Contents.....	3
Terminology, Definitions, Abbreviations.....	4
1. Introduction.....	5
2. ReqIF and data roundtrip	5
2.1 ReqIF use cases.....	5
3. ReqIF Semantics implementation background	6
3.1 Data exchange process	6
3.2 Schemas and schema languages.....	7
3.3 XSD schema: datatypes, components and references.....	8
3.4 Possible approaches	12
3.5 Technologies	13
3.6 Semantics and ontologies.....	15
4. Demonstration of SIMANTICS ReqIF capability	15
5. Conclusions	20
References.....	21

Terminology, Definitions, Abbreviations

Abbreviation	Description
DOM	Document Object Model
DTD	Document Type Definition
JAXB	Java™ Architecture for XML Binding
JAXP	Java™ API for XML Processing
JDK	Java™ Development Kit
OMG	Object Management Group
OSGi™	The Dynamic Module System for Java™
PSVI	Post Schema Validation Infoset
RDF	Resource Document Format
Relax NG	A simple schema language for XML, based on RELAX and TREX.
ReqIF	Requirements Interchange Format
RM	Requirement Management
SAX	Simple API for XML
Schematron	Document Schema Definition Language (ISO/IEC 19757-3, 2006)
StAX	Streaming API for XML
TrAX	Transformations API for XML
W3C	World Wide Web Consortium
XML	eXtensible Markup Language
XMLBeans	A technology for accessing XML by binding it to Java types
XNI	Xerces Native Interface
XSD	XML Schema Definitions

1. Introduction

This report focuses on the steps to support ReqIF data roundtrip and data validation in Simantics environment. Chapter 2 introduces ReqIF and a use case to exchange ReqIF data between two companies. The use case scenario serves as a background idea for the Simantics ReqIF implementation. Firstly, the Chapter 3 generalizes a problem statement from ReqIF data exchange to general XML support in the Simantics environment. Also, the scope of research is set up. The data exchange process is presented in terms of Simantics. Then, it is explained what is purpose of schemas and what they are. Usefulness of different schema languages is estimated. Schematron, DTD, XML Schema and RELAX NG are under consideration. Study continues deeper with XML Schema, supported data types and abstract data model behind XML Schema (i.e. schema component model). Approaches to exchange XML data with Simantics are introduced and technological possibilities are evaluated. In the Chapter 4, main features of the Simantics ReqIF implementation, based on selected approach and technology, are demonstrated and documented. Finally, the Chapter 5 contains conclusions.

2. ReqIF and data roundtrip

ReqIF¹ is an XML (XML 1.1, 2006) specification for requirements interchange. The ReqIF specification is maintained by Object Management Group (OMG) (ReqIF 2013). The goal of ReqIF is to provide a file format to exchange requirements between different organisations or departments of an organisation developing a system. The file format is based on XML. The ReqIF specification is comprehensive enough to support different brands of requirements management tools on the market to facilitate cooperation of organisations with different tools. A common case is in which a system developer passes requirements to its subcontractors; the subcontractor may update the requirements or comment them and may send the updated set of requirements in ReqIF format back to the main contractor.

ReqIF allows exchange of all kinds of artefacts (and their attributes) besides requirement statements. This gives the possibility to pass verification and validation information, like simulation results, back from the subcontractors to the requirements originator.

ReqIF is not a model for the requirements management database structure. However, there is a tool on the market that allows editing of ReqIF files natively. The tool is called ProR. ProR can be used as a simple requirements authoring tool, but it does not support version control.

2.1 ReqIF use cases

Figure 1 depicts the procedure of roundtrip exchange of requirements between two parties with different or same requirements management tools (RM tool). One important question is raised by the roundtrip use case scenario: How well the RM tool A is able to synchronise its database with the ReqIF XML file received from RM tool B? There may be new requirements, changes in original requirements and deletions of original requirements by User/Company 2, but also by User/Company 1 during the User/Company 2 work if no concurrency control is exercised. This question is also raised in case of a one-way scenario (in which User/Company 2 only receives ReqIF files and does not send such back): How well the RM tool B is able to synchronise its database if User/Company 1 sends an updated ReqIF file?

¹ Formely known as RIF, Requirements Interchange Format.

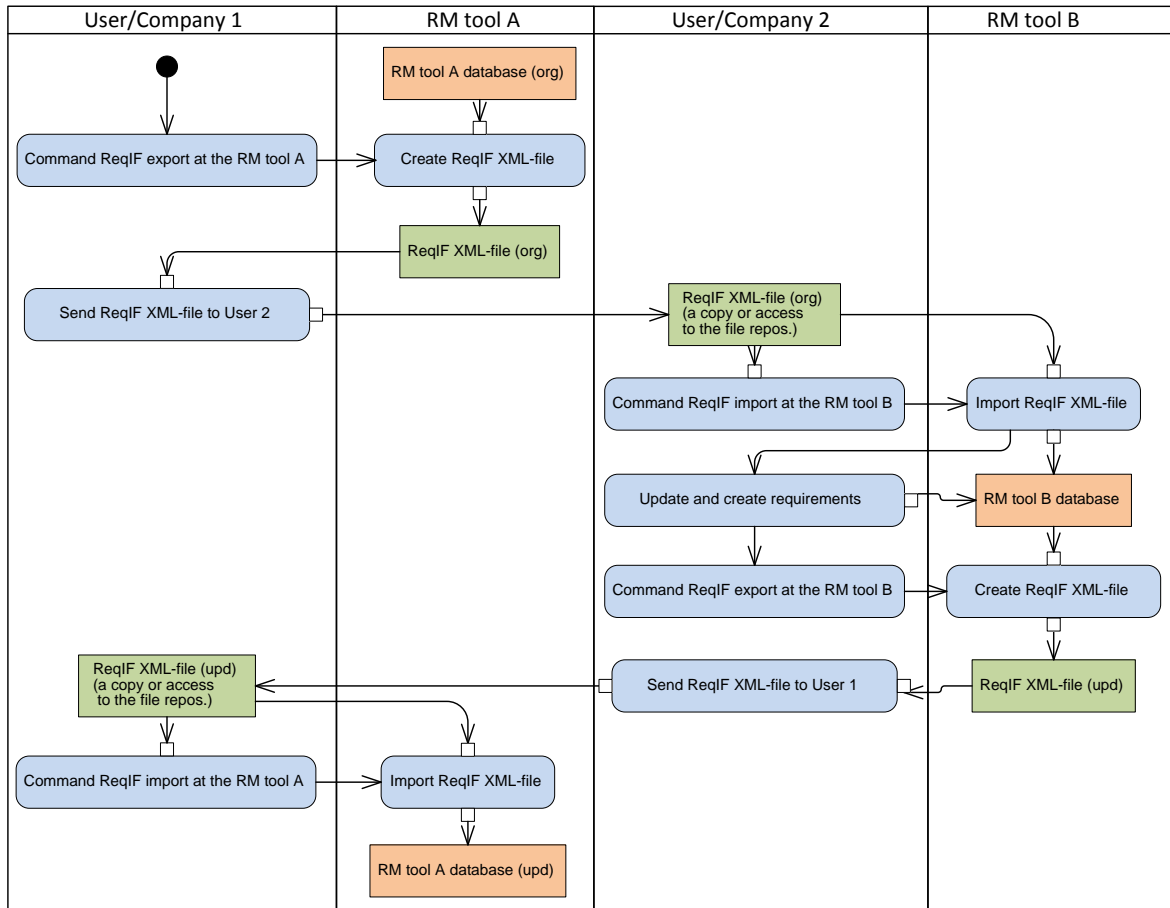


Figure 1. ReqIF roundtrip use case scenario.

Poor implementation of the synchronisation makes the ReqIF interchange very impractical. Hence usage of a shared requirements database is a considerable alternative over ReqIF file interchange.

A detailed description of the one-way and roundtrip scenarios can be found in the ReqIF (2013) specification.

3. ReqIF Simantics implementation background

3.1 Data exchange process

An approach to demonstrate ReqIF data exchange with the Simantics (Simantics, 2014) integration platform is presented in Figure 2.

Prerequisite to transfer XML data consistently is depend on data validation possibilities. In this respect, schemas have an important role in quality assurance and they are used for schema-validity assessment of XML information. According to this observation the Simantics data exchange approach has been divided to the following phases:

- Import schema files to support ReqIF data in Simantics
- Export ReqIF data from another tool (not present in Figure 2)
- Import ReqIF data into Simantics. The data is validated and bound to concepts provided by the schema while importing.
- Manipulate data in the Simantics environment or transform it totally to another form with the Simantics Constraint Language (SCL)

- Export updated data from Simantics in XML format in order to support data round trip between a system developer and a subconactor.

In typical business cases, data round trip operations occur multiple times for same data. This means data import over existing data that can be supported by data comparison or versioning as explained in the first chapter. This kind of functionality is not yet supported in Simantics because common versioning mechanism (change sets) is under design by the Simantics core developers at this time. In addition, it is clear that a friendly user interface is needed to manipulate requirements and related data but ergonomic graphical user interface design is currently out of scope of this research. Thus, this report focuses on the first steps to support ReqIF data roundtrip and data validation in the Simantics environment.

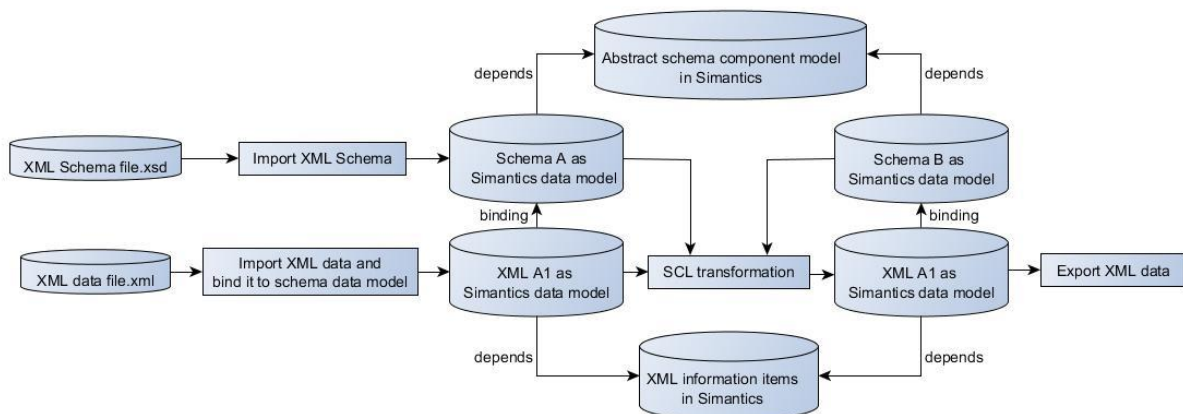


Figure 2. ReqIF data exchange approach with Simantics.

3.2 Schemas and schema languages

Schemas have an important role in a validation of XML documents and in XML based application development. According to Murata (Murata et al., 2001) a schema specifies permissible names for XML elements and attributes, and further specifies permissible structures and values for these elements and attributes. Schema creation can provide the following advantages:

1. The schema precisely describes permissible XML documents
2. A computer programs can determine whether or not a given XML document is permitted by the schema,
3. A schema can be used in application program creation (by generating code skeletons, for example).

There exist several languages for specifying schemas and they are called schema languages. A schema language can be based on general XML constructs (as in the case of DTD based on elements and attributes) or more specific constructs representable by XML (as in the case of RDF Schema based on resources, properties and statements). On the other hand, Eric van der Vlist (van der Vlist, 2001) describes XML schema languages in the following way: "All XML schema languages define transformations to apply to a class of instance documents. XML schemas should be thought of as transformations. These transformations take instance documents as input and produce a validation report, which includes at least a return code reporting whether the document is valid and an optional Post Schema Validation Infoset (PSVI), updating the original document's infoset (the information obtained from the XML document by the parser; XML InfoSet, 2004) with additional information (default values, datatypes, etc.)". This description concerns with validation process which is also a major key in our implementation and data interpretation (or data typing).

There are many kind of comparisons related to features of different schema languages (XML Schema languages, 2014; Dongwon et al., 2000). Comparison of Eric van der Vlist (van der Vlist, 2001) is based on validation levels enabled by schema languages. These levels are:

- The validation of markup – controlling the structure of a document
- The validation of the content of individual leaf nodes (datatyping)
- The validation of integrity, i.e. the validation of the links between nodes within a document or between documents.
- Any other tests (often called "business rules").

The levels define important decision points while selecting a schema language for the ReqIF Simantics implementation. From design point of view a schema language is required to support data interpretation in order to make data typing while importing XML document into Simantics. In addition, validation of document structure and integrity are relevant aspects while creating ontology based information model correctly.

Although, the Document Type Definition (DTD) (in XML 1.0, 2008) is widely used and one of easiest schema languages it is not selected for the ReqIF Simantics implementation because it provides only a restricted content model specifications for XML elements. On the other hand, Relax NG (RELAX NG, 2001) provides a lot of expression power for element content specifications but an augmentation is not supported. This means that Relax NG can natively not provide default values for element attributes nor typing information or PSVI. Thus, the creation of ontology information model will lead to incomplete model. Thirdly, Schematron schemas (ISO/IEC 19757-3, 2006; Jelliffe, 2001) or rule based schemas lead typically very verbose specification. For that reason, Schematron schemas are often used together with W3C XSD (XML Schema Definitions) or RELAX NG to define only the most specific constraints for XML document contents. Usage of Schematron means hybrid model in design and may lead to over complicated implementation and hard maintenance. For these reasons, the ReqIF Simantics implementation avoids the usage of Schematron. Thus, W3C XSD is selected as the most suitable schema language for the purpose in the ReqIF Simantics implementation. In addition, some XML parsers can provide a consistent Post Schema Validation Infoset based on abstract schema component model of W3C XSD during XML processing.

3.3 XSD schema: datatypes, components and references

W3C (XSD 1.1) XML Schema Definitions are defined in three parts in the W3C XML Schema specifications. XML Schema Part 0 (XML Schema, 2004) is an introductory document (primer), XML Schema Part 1 (XSD 1.1, 2012a) offers facilities for describing the structure and constraining the contents of XML documents and XML Schema Part 2 (XSD 1.1, 2012b) defines facilities for defining datatypes to be used in XML Schemas as well as in other XML specifications. The datatype language provides a superset of the capabilities found in DTDs.

W3C XSD is used to provide a vocabulary (list of elements and attributes), to associate types (such as integer, string, hatsize, sock_colour, etc.) with element and attribute content, to constrain appearance of elements and attributes, to provide human-readable and machine-processable documentation and to give a formal description of one or more documents.

To ensure robustness in document interpretation and data interchange a high degree of type checking is required. XML Schema Part 2 defines **built-in datatypes** that can be used in an XML Schema. A datatype is characterized by its value space, lexical space and lexical mapping. The lexical mapping is a relation which maps the lexical space of datatype into its value space. The specification defines atomic, list and union datatypes. **Atomic value** is an elementary value, not constructed from simpler values by any user-accessible mean of the specification. Despite of this the values may be described with internal structure, which can be utilized while checking value constraints. **Atomic datatypes** are those whose value spac-

es contain only atomic values. Values of **list datatypes** consist of a finite-length (possibly empty) sequence of atomic values separated by a space and specified by item type attribute of the list datatype. Thus, a lexical representation of a list element can't contain a whitespace. **Union datatypes** are those whose value spaces, lexical spaces, and lexical mappings are the union of the value spaces, lexical spaces, and lexical mappings of one or more other datatypes or those that are derived by facet-based restriction of another union datatype. The datatypes which participate to union datatype declaration are called to member types of the union.

In addition, the specification introduces the following facilities for new datatype generation.

- facet-based restriction
- construction by list
- construction by union

The facet-based restriction constructs a new datatype by restricting the value space or lexical space of a base type using zero or more constraining facets (e.g. pattern, enumeration, max-Inclusive, minExclusive). Construction by list defines a new datatype by specifying the new datatype as a list of items of some item type and construction by union specifies a new datatype by defining it as a union of some specified sequence of member types. The specification uses also these facilities to demonstrate the provided mechanisms. Figure 3 presents build-in datatypes which are described as an independent of their use in the particular context.

Build-in datatypes are distinguished to special, primitive and ordinary (or constructed) datatypes. Datatypes `anySimpleType` and `anyAtomicType` are special according to their position in the type hierarchy. Primitive datatypes are those datatypes that are not special and are not defined in terms of other datatypes. All primitive datatypes have `anyAtomicType` as their base type, but their value and lexical spaces are given in prose. Ordinary datatypes are all datatypes other than the special and primitive datatypes. User-defined datatypes are constructed with Simple Type Definition schema component by schema designer. The mechanism defines a datatype in terms of other datatypes and attaches a qualified name (QName) to it. More generally, the mechanism integrates build-in datatypes into XML XSD context because the given datatypes of the specification are also intended to be useful in other contexts.

The XML Schema Part 1 (Structures) specification offers facilities for describing the structure and constraining the contents of XML documents. In addition, extra information for an XML document or a corresponding XML Infoset augmentation, such as normalization and defaulting of attribute and element values or the types of the information items can be specified. The XML Schema Part 1 introduces a conceptual and abstract data model for XML Schema information which must be available in conformant processing but the specification does not mandate any particular implementation or representation of the data model information. The abstract data model has been made up from schema components and thus an XSD schema is defined as a set of schema components. According to the specification the schema components are falling into groups:

- Primary schema components
 - Simple type definitions
 - Complex type definitions
 - Attribute declarations
 - Element declarations
- Secondary schema components
 - Attribute group definitions
 - Identity-constraint definitions
 - Type alternatives
 - Assertions

- Model group definitions
- Notation declarations
- "Helper" schema components
 - Annotations
 - Model groups
 - Particles
 - Wildcards
 - Attribute Uses

The schema components named with "definition" end define schema components that are used by other schema components. The helper components are dependent on their context. The declaration components are associated by (qualified) name to XML information items being validated during validation process. Many components have a target namespace, which is either absent or a namespace name (XML Namespaces, 2006). The target namespace identifies the namespace within which the association between the component and its name exists. The target namespace for predefined schema components is <http://www.w3.org/2001/XMLSchema>. Detailed descriptions of schema components, their properties, XML representations, their contributions to post-schema-validation infoset (PSVI) and validation can be found from the specification.

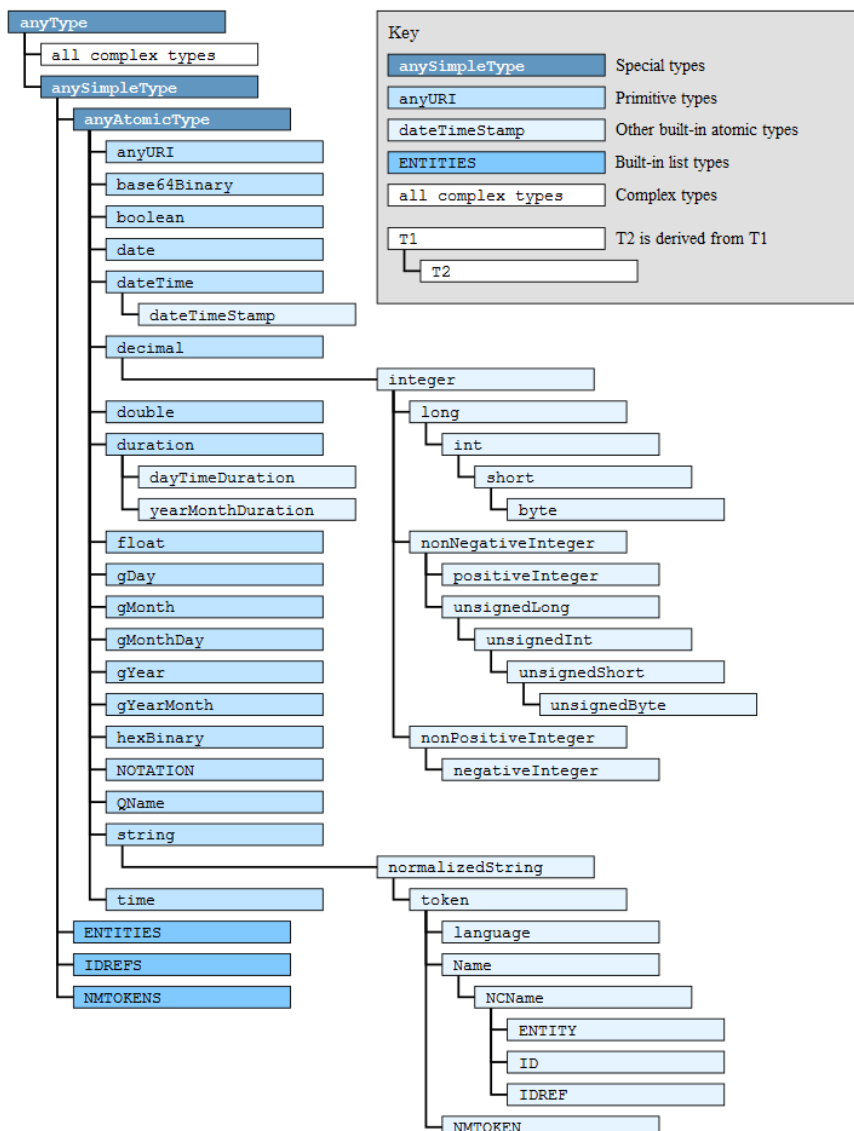


Figure 3. Build-in datatypes in XML XSD.

It has to pay attention to a difference of XSD Schema and schema document. XSD Schema corresponding to a schema document contains definitions and declarations of the schema document, built-in schema components, automatically known components (primitive or user defined) and schema components included by consequence of the inter-schema-document references. Thus, XSD Schema is assembled from multiple sources and can contain a lot of schema components (with name association from different namespaces) not defined in a schema document. On the other hand, the schema document can define at most one (or none) **targetNamespace** associating top-level schema components of that document with a target namespace. To license references to components outside the document's target namespace (to components in the imported namespace) the `<import>` element is used in schema document.

```
<xs:import namespace="anyURI" schemaLocation ="anyURI"/>
```

The location of schema document for imported namespace can also be provided as `schemaLocation` attribute value. The `schemaLocation` operates only as a hint which processor may attempt to de-reference. If namespace attribute is absent, then the import allows unqualified reference to components with no target namespace.

Secondly, `<include>` element is used to assemble schema from multiple schema documents to single target namespace. For example,

```
<xs:include schemaLocation ="anyURI"/>
```

The included schema documents must either have the same `targetNamespace` as referencing schema document or no `targetNamespace` at all. In the latter case, the included schema components are converted to target namespace of assembling schema document.

Thirdly, `<override>` element is used to replace old schema components with new ones without any constraint. For example,

```
<xs:override schemaLocation="v1.xsd">
  <xs:complexType name="personName">...
</xs:override>
```

replaces the "personName" `complexType` definition located in `v1.xsd` schema document by a new one. For `<include>` and `<override>` elements the processor must attempt to de-reference according to a value of the `schemaLocation` attribute. Thus, the elements `<import>`, `<include>` and `<override>` signal a schema-aware processor that a schema document contains references to other schema documents.

More fine grained control to include certain schema components to a schema according to version of schema specification or types known by processor is presented as condition inclusion attributes. These attributes are `vc:minVersion`, `vc:maxVersion`, `vc:typeAvailable`, `vc:typeUnavailable`, `vc:facetAvailable` and `vc:facetUnavailable`.

For validation a XML instance document can provide hints to a processor about relevant schema document locations. The instance document refers to schema document locations by `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attribute values.

```
<purchaseReport
  xmlns="http://www.example.com/Report"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.com/Report
  http://www.example.com/Report.xsd"
```

The `schemaLocation` attribute value consists of one or more pairs of URI references. The first part is a namespace name, and the second is a hint describing where to find an appro-

ropriate schema document for the namespace. The processor is free to use the referenced schema documents or other schemas or to use no schema at all. If a schema document does not define the `targetNamespace` the instance document refers to this kinds of schema documents by `noNamespaceSchemaLocation` attribute. In this case, the attribute lists only schema location URIs.

3.4 Possible approaches

To support requirement management and exchange of ReqIF requirements in the Simantics platform (Simantics), several approaches and technologies have been studied. The following approaches have been conceived as possible ways to enable ReqIF based requirement management in Simantics:

- JAXB (JSR 222, 2009) based XML information binding
- XMLBeans (XMLBeans, 2012) based XML information binding
- JAXP (JSR 206, 2013), Java™ API for XML Processing
- DOM (DOM 3, 2004) based XML-document handling
- Simple API for XML (SAX 2, 2002) based XML-document handling
- StAX (JSR 173, 2014) based XML-document handling
- TrAX (TrAX, 2006), Transformations API for XML, Apache Xalan

Before pros and cons of the approaches are presented the research problem is generalized. From the Simantics view point a challenge is: How any kind of XML information can be exchanged between Simantics and other applications. Thus, the generalized goal is to demonstrate that data from heterogeneous XML sources, actually files, can be collected to Simantics and the ReqIF data is considered as a special use case. This kind of generalization is justified because Simantics is an integration platform and more benefits can be reached by the generalized implementation. In addition, it is required that XML data can be validated in order to avoid missing or misleading data fragments in data exchange. This means that XML data has also to have XML schema declarations.

XML information mapping or binding approaches based on JAXB or XMLBeans requires that XML schema definitions (XSD 1.1 2012a) are available and their definitions can be compiled to corresponding Java classes. Therefore, these approaches can work quite efficiently with ReqIF requirements but not in the context of the generalized problem. In the latter case, schema and Java compilers have to be integrated into Simantics in order to work with other kind of XML data and XML schemas. In addition, JAXB supports only partly XML Schema 1.0 definitions. XMLBeans has a full XML Schema 1.1 support.

JAXP combines a number of Java APIs and enables applications to parse and transform XML documents using implementation independent APIs. The plugability layer of JAXP includes Datatype, XPath, XSLT (TrAX), Validation, StAX, SAX and DOM plugabilities. The plugability mechanism allows to dynamically load compliant implementations or to switch between particular XML processor implementations. (JSR 206, 2013; JAXP 1.4)

Typically, DOM based approaches handle XML documents in memory representation. It provides a tree-based API to handle XML data. These kinds of APIs are quite friendly to programmers because XML data can be traversed by API or the needed data can be queried by XQuery language (XQuery 3.0, 2014) or XPath (XPath 3.0, 2014) expressions. In addition, all information is virtually available all the time and DOM API can provide features to validate XML data. However, DOM based approach is not selected for the Simantics data exchange because this kind of technologies do not provide enough computation power for large XML files.

SAX and StAX are event-based APIs for XML data processing. They provide XML information by reporting parsing events to handlers registered by an application. Difference be-

tween APIs is that StAX provides pull-parsing API and SAX provides push-parsing API. In other world, StAX gives parsing control to programmer while SAX takes the program control during parsing. Both APIs are very powerful with large XML files because there is no need to construct internal data representation as in the case of DOM based approach. In addition, both event-based APIs consume only a little memory resources and they can support XML data validation with XML schemas

TrAX consists actually of multiple APIs. A challenge of the TrAX is how to deal with different kind of inputs and outputs without becoming specialized for any types. Inputs and outputs may have form of URL, XML stream, a DOM tree, SAX Events, or a proprietary format or data structure. In addition, transformations can be described by Java code, Perl code, XSLT Stylesheets, other types of script or even in proprietary formats. TrAX is not studied for the Simantics data exchange although it can be used to specify and execute XML transformations with XSLT engines.

In conclusion, the several possible approaches have been proposed to exchange ReqIF requirements with Simantics. Some of them have limitations in XML schema support and others in their data processing performance. In addition, some can lead to over complicate architecture. Finally, SAX and StAX based approaches have been evaluated and selected as the most suitable to overtake stated goals and to satisfy the presented requirements.

3.5 Technologies

This chapter describes some useful technologies for the most attractive implementation approach. The following possible and interesting technologies have been identified:

- Apache Xerces based implementation
- JDK based implementation
- Woodstock based implementation

In addition, the role of XML OASIS catalogues (XML Catalogs, 2005) has to be taken into account while implementing the approach.

XML catalogues are entity catalogues that map both external identifiers and arbitrary URI references to URI references for desired resources. With OASIS standard the following cases can be handled.

- Mapping an external entity's public identifier and/or system identifier to a URI reference.
- Mapping the URI reference of a resource (a namespace name, stylesheet, image, etc.) to another URI reference.

By using XML Catalogs the several XML Schema files can be pre-downloaded and references to these files can be redirected while parsing XML information. This kind of configuration reduces parsing time a lot because download times are saved. OASIS XML Catalog manager looks for `CatalogManager.properties` file which defines locations of XML Catalogs.

```
CatalogManager.properties
catalogs=./CatalogForResolver.xml
```

Below is also an example of a content of catalogue file which maps system identifiers to local file URI references.

CatalogForResolver.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog SYSTEM "http://www.oasis-open.org/committees/entity/release/1.0/catalog.dtd" PUBLIC "-//OASIS//DTD Entity Resolution XML Catalog V1.0//EN">
<catalog prefer="public" xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <system uri="file:///C:/MathML/mathml3-common.xsd" systemId="mathml3-common.xsd"/>
  <system uri="file:///C:/MathML/mathml3-strict-content.xsd" systemId="mathml3-strict-content.xsd"/>
  <system uri="file:///C:/MathML/mathml3-presentation.xsd" systemId="mathml3-presentation.xsd"/>
  <system uri="file:///C:/MathML/mathml3-content.xsd" systemId="mathml3-content.xsd"/>
</catalog>
```

Apache Xerces2 Java Parser 2.11.0 (Xerces2, 2010) supports the following standards and APIs:

- eXtensible Markup Language (XML) 1.0 Fourth Edition Recommendation
- Namespaces in XML 1.0 Second Edition Recommendation
- eXtensible Markup Language (XML) 1.1 Second Edition Recommendation
- Namespaces in XML 1.1 Second Edition Recommendation
- XML Inclusions (XInclude) Version 1.0 Second Edition Recommendation
- Document Object Model (DOM) Level 3 Core, Load and Save, (DOM) Level 2 Core, Events, and Traversal and Range Recommendations
- Element Traversal First Edition Recommendation
- Simple API for XML (SAX) 2.0.2 Core and Extensions
- Java APIs for XML Processing (JAXP) 1.4
- Streaming API For XML (StAX) 1.0 Event API (javax.xml.stream.events)
- XML Schema 1.0 Structures and Datatypes Second Edition Recommendations
- XML Schema 1.1 Structures and Datatypes Working Drafts (December 2009)
- XML Schema Definition Language (XSD): Component Designators (SCD) Candidate Recommendation

Its latest version can be downloaded from

```
http://xerces.apache.org/mirrors.cgi
Xerces Java 2.11.0 (XML Schema 1.1) (Beta)
```

Xerces2 is promising to achieve the goal because the newest version (v2.11) introduces the Xerces Native Interface (XNI) which enables a complete framework for building parser components and configurations in extremely modular and easy way. The framework provides also support for OASIS XML Catalogs and defines an XML Schema API (Litani, 2004) for accessing and querying the post schema validation infoset (PSVI). In addition, the API defines interfaces for loading XML schema documents (javax.xml.validation.SchemaFactory) and for accessing a generated data model of schema or a set of schema components through XML Schema Component API. For event based XML document processing the javax.xml.parsers.SAXParserFactory API enables applications to configure and obtain a SAX based parser. Sometimes, it is challenging to know the actual SAXParserFactory implementation class resolved according to JAXP. In these kinds of situations the following command can be aid of:

```
>java -Djaxp.debug=1 YourProgram
```

XML parsers are also provided by Java Development Kit (JDK). JDK includes typically an older Xerces version as Apache can provide. To check the version of Xerces bundled with the JDK you can type a command in command prompt (<http://cafe.elharo.com/xml/what-version-of-xerces-are-you-using/>, Referenced 1 Sep 2015):

```
> java com.sun.org.apache.xerces.internal.impl.Version
```

The command can return e.g. Xerces-J 2.7.1. Because the older Xerces versions neither implement XML Schema API nor grammar pool features the JDK enabled parser is not used in the Simantics data exchange.

Woodstock (Woodstox, 2015) is a high-performance Java XML processor. It implements Streaming XML API (JSR-173, 2014), and supports XML document validation with DTD, W3C Schema or RelaxNG. It is available under two open source licenses (Apache License, LGPL). However, Woodstock is limited in term of schema caching features.

3.6 Simantics and ontologies

The Simantics platform (Simantics, 2014) is an open and high level application platform on which different computational tools can be easily integrated to form a common environment for modelling and simulation. Simantics has semantic modelling features and high-level ontology tools. A high performance data management and arbitrary data mappings are responsible of a data triple engine on the server side. Simantics is based on OSGi plugin architecture (OSGi, 2015) and is implemented by Eclipse (Eclipse, 2015). Thus, Eclipse plugins are actually used.

In Simantics, ontology is a compilation of shared concepts (types, relations and instances). The ontologies are defined by a graph file format. A graph file describes a collection of statements. One statement is written as a triple: subject-predicate-object. Subject, predicate and object are all resources in the Simantics platform. Multiple statements for a subject can be written by presenting only new consecutive predicate-object pairs. In addition, statements for fixed subject and predicate can be abbreviated by presenting only new objects. Below is an example which defines new movie instance (tt0381061) under library resource.

```
MO = <http://www.acme.com/Movie-1.0>
ML = <http://www.acme.com/MovieLibrary> : L0.Library
    @L0.new

ML.tt0381061 : MO.Movie
    MO.HasTitle "Casino Royale" : L0.String
    MO.IsDirectedBy ML.MartinCampbell
    @MO.Casting ML.JamesBond ML.DanielCraig
```

HasTitle and IsDirectedBy predicates are relations. HasTitle refers to a literal resource with value "Casino Royale" and IsDirectedBy refers to another resource under the ML which is identified by MartinCampbell name. The @MO.Casting reminds the usage of a template. More information can be found from the Simantics Developer Documentation and its Ontology Development section (Simantics, 2015).

4. Demonstration of SIMANTICS ReqIF capability

Implementation of ReqIF for the Simantics platform utilizes ontologies and data triple engine of Simantics. To enable ReqIF data exchange (Figure 1) and requirement management Simantics has firstly to recognize the XML Schemas in some way (Figure 2). This is achieved by predefining the schema component model (introduced in the Section 3.3) as ontology of Simantics (Simantics, 2015). The OSGi plugin named

```
org.simantics.xml.schema.ontology
```

contains these schema component definitions which form a base for implementation. Below are example definitions for the ElementDeclaration schema component.

```
SCC.SchemaComponent <T L0.Entity
-- SCC.SchemaComponent.targetNamespaceOf <R L0.IsRelatedTo
--> SCC.SchemaComponent
```



```

>-- SCC.SchemaComponent.targetNamespace <R L0.IsRelatedTo L0.InverseOf
SCC.SchemaComponent.targetNamespaceOf
--> L0.Library
L0.HasDescription "Target namespace of schema.":L0.String

SCC.Term <T SCC.SchemaComponent

SCC.ElementDeclaration <T SCC.Term
>-- SCC.ElementDeclaration.annotations <R L0.IsComposedOf : L0.FunctionalRelation
--> L0.List
L0.HasDescription "A sequence of Annotation components.":L0.String
>-- SCC.ElementDeclaration.name <R L0.HasProperty : L0.FunctionalRelation
--> L0.String
L0.HasDescription "An xs:NCName value. Required.":L0.String
>-- SCC.ElementDeclaration.targetNamespace <R L0.HasProperty : L0.FunctionalRelation
--> L0.String
L0.HasDescription "An xs:anyURI value. Optional.":L0.String
>-- SCC.ElementDeclaration.typeDefinition <R L0.IsRelatedTo : L0.FunctionalRelation
--> SCC.TypeDefinition
L0.HasDescription "A Type Definition component. Required.":L0.String
>-- SCC.ElementDeclaration.typeTable <R L0.HasProperty : L0.FunctionalRelation
--> SCC.TypeTable
>-- SCC.ElementDeclaration.scope <R L0.HasProperty : L0.FunctionalRelation
--> SCC.Scope
L0.HasDescription "A Scope property record. Required.":L0.String
>-- SCC.ElementDeclaration.valueConstraint <R L0.HasProperty : L0.FunctionalRelation
--> SCC.ValueConstraint
L0.HasDescription "A Value Constraint property record. Optional.":L0.String
>-- SCC.ElementDeclaration.nullable <R L0.HasProperty : L0.FunctionalRelation
--> L0.Boolean
L0.HasDescription "An xs:boolean value. Required.":L0.String
>-- SCC.ElementDeclaration.identityConstraintDefinitions <R L0.IsRelatedTo
--> SCC.IdentityConstraintDefinition
L0.HasDescription "A set of Identity-Constraint Definition components.":L0.String
>-- SCC.ElementDeclaration.substitutionGroupAffiliations <R L0.IsRelatedTo
--> SCC.ElementDeclaration
L0.HasDescription "A set of Element Declaration components.":L0.String
>-- SCC.ElementDeclaration.substitutionGroupExclusions <R L0.IsRelatedTo
--> SCC.ElementDeclaration.SubstitutionGroupExclusionEnum
L0.HasDescription "A subset of {extension, restriction}.":L0.String
>-- SCC.ElementDeclaration.disallowedSubstitutions <R L0.IsRelatedTo
--> SCC.ElementDeclaration.DisallowedSubstitutionsEnum
L0.HasDescription "A subset of {substitution, extension, restriction}.":L0.String
>-- SCC.ElementDeclaration.abstract <R L0.HasProperty : L0.FunctionalRelation
--> L0.Boolean
L0.HasDescription "An xs:boolean value. Required.":L0.String

```

To import ReqIF schema or any other schemas into Simantics, the plugin

```
org.simantics.xml.ui
```

defines a command

```
org.simantics.xml.ui.schemaXniImport
```

and a command handler

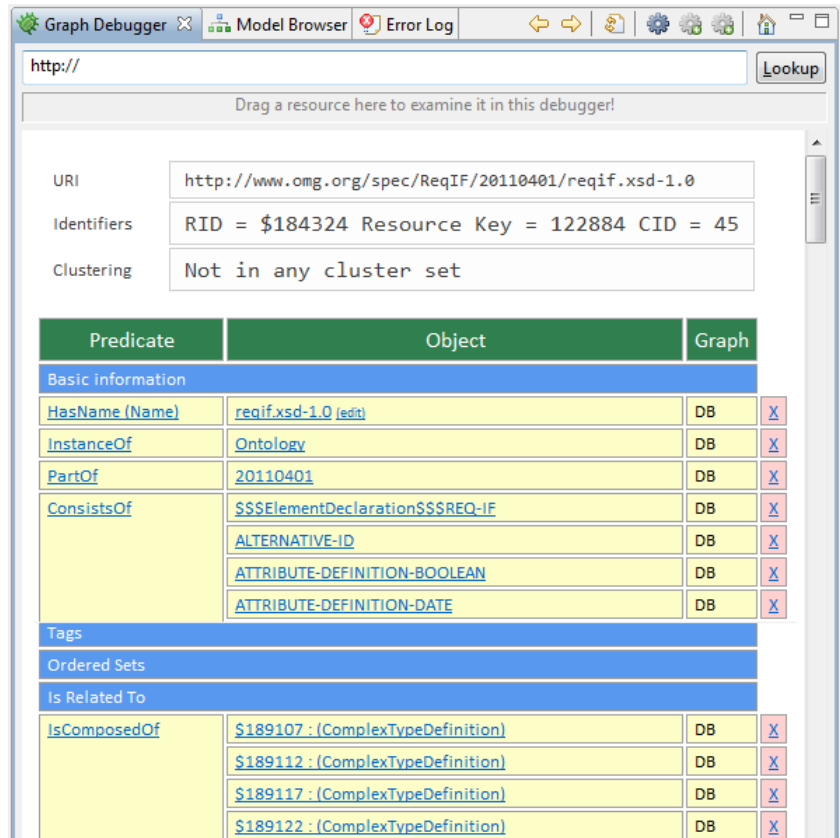
```
org.simantics.xml.ui.browser.handler.SchemaXniImport.
```

for that purpose. The implementation is based on Apache Xerces2 Java Parser 2.11.0 (Section 3.5; Xerces2, 2010) and a feature of Xerces Native Interface (XNI) framework which provides an interface

```
org.apache.xerces.xni.grammars.XMLGrammarPool
```

for an application to interact with a parser to exchange grammar objects or XML Schemas and XML Schema components. The feature gives a possibility to the application to store or cache the grammars. In XML document validation the parser requests initial grammars at start up by `retrieveInitialGrammarSet` method. If a needed grammar is not in an initial set the parser requests it by `retrieveGrammar` method. After successfully validating an XML

document, the validator makes any new grammars available to application by using the `cacheGrammars` method. The classes `org.simantics.xml.schema.xni.GrammarPool` and `org.simantics.xml.schema.xni.psvi.SSGrammarPool` implement grammar pooling into the Simantics database. The breadth-first-search algorithm is used to cache grammars as Simantics ontologies. Figure 4 shows snapshots of imported and cached ReqIF schema in the Simantics database.



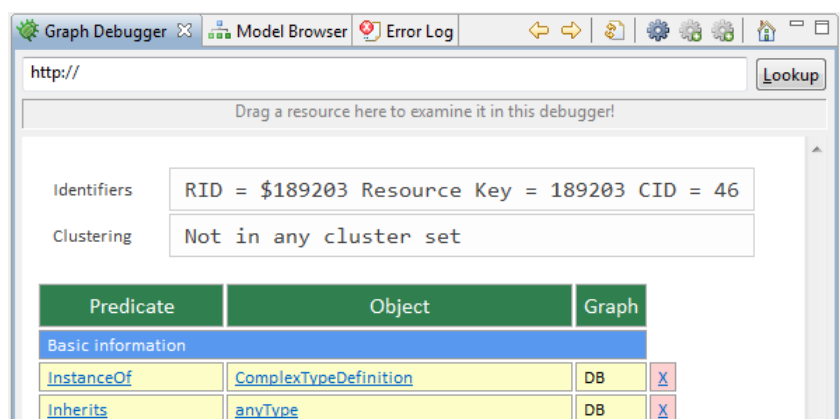
The screenshot shows the Graph Debugger interface with the following details:

- URI: `http://www.omg.org/spec/ReqIF/20110401/reqif.xsd-1.0`
- Identifiers: RID = \$184324 Resource Key = 122884 CID = 45
- Clustering: Not in any cluster set

Predicate	Object	Graph
Basic information		
HasName (Name)	reqif.xsd-1.0 (edit)	DB X
InstanceOf	Ontology	DB X
PartOf	20110401	DB X
ConsistsOf	\$\$\$ElementDeclaration\$\$\$REQ-IF	DB X
	ALTERNATIVE-ID	DB X
	ATTRIBUTE-DEFINITION-BOOLEAN	DB X
	ATTRIBUTE-DEFINITION-DATE	DB X
Tags		
Ordered Sets		
Is Related To		
IsComposedOf	\$189107 : (ComplexTypeDefinition)	DB X
	\$189112 : (ComplexTypeDefinition)	DB X
	\$189117 : (ComplexTypeDefinition)	DB X
	\$189122 : (ComplexTypeDefinition)	DB X

Figure 4. ReqIF Schema imported into Simantics.

Several observations can be seen from the snapshots. URI fragments of the target namespace of the schema are used to form a hierarchy into the Simantics database. In addition, different kind of schema components can have same name in the schema. This is not allowed in Simantics. Thus, some prefixes are used (e.g. `$$$ElementDeclaration$$$`) to differentiate the schema components. Figure 5 presents a complex type definition in Simantics. This complex type definition has no name and it has been inherited from a predefined any-type definition.



The screenshot shows the Graph Debugger interface with the following details:

- Identifiers: RID = \$189203 Resource Key = 189203 CID = 46
- Clustering: Not in any cluster set

Predicate	Object	Graph
Basic information		
InstanceOf	ComplexTypeDefinition	DB X
Inherits	anyType	DB X

Figure 5. Unnamed complex type definition.

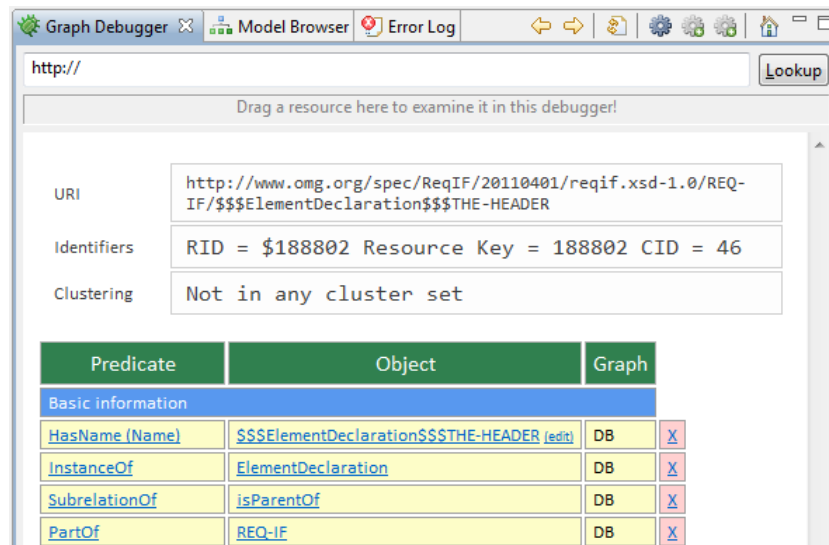


Figure 6. Element declaration.

Figure 6 shows THE-HEADER element declaration of the ReqIF schema in Simantics. It is declared under REQ-IF complex type definition and is an instance of ElementDeclaration. One interesting notice is that schema component element declarations are modelled as relations in Simantics unlike schema component attribute declarations which are modelled as properties (SubrelationOf HasProperty) in Simantics.

To import ReqIF requirements the plugin

```
org.simantics.xml.ui.browser.handler
```

defines a command

```
org.simantics.xml.ui.xmlXniImport
```

and a command handler

```
org.simantics.xml.ui.browser.handler.XmlXniImport
```

for that purpose. XML data is imported according to XML InfoSet data model (XML InfoSet, 2004). The XML data model has been defined as ontology and it is predefined in the plugin

```
org.simantics.xml.ontology
```

The ontology contains XML information item definitions. Below is a definition for the ElementInfoItem.

```
XIS.ElementInfoItem <T XIS.InfoItem
  >-- XIS.ElementInfoItem.namespaceName <R L0.HasProperty : L0.FunctionalRelation
  --> L0.String
  L0.HasDescription "The namespace name, if any, of the element type. If the element
  does not belong to a namespace, this property has no value.":L0.String
  >-- XIS.ElementInfoItem.localName <R L0.HasProperty : L0.FunctionalRelation
  --> L0.String
  L0.HasDescription "The local part of the element-type name. This does not include any
  namespace prefix or following colon.":L0.String
  >-- XIS.ElementInfoItem.prefix <R L0.HasProperty : L0.FunctionalRelation
  --> L0.String
  L0.HasDescription "The namespace prefix part of the element-type name. If the name is
  unprefix, this property has no value. Note that namespace-aware applications should
  use the namespace name rather than the prefix to identify elements.":L0.String
  >-- XIS.ElementInfoItem.children <R L0.IsComposedOf : L0.FunctionalRelation
  --> L0.List
```

```
L0.HasDescription "An ordered list of child information items, in document order. This list contains element, processing instruction, unexpanded entity reference, character, and comment information items, one for each element, processing instruction, reference to an unprocessed external entity, data character, and comment appearing immediately within the current element. If the element is empty, this list has no members.":L0.String
>-- XIS.ElementInfoItem.attributes <R L0.HasProperty
--> XIS.AttributeInfoItem
L0.InverseOf XIS.AttributeInfoItem.ownerElement
L0.HasDescription "An unordered set of attribute information items, one for each of the attributes (specified or defaulted from the DTD) of this element. Namespace declarations do not appear in this set. If the element has no attributes, this set has no members.":L0.String
>-- XIS.ElementInfoItem.namespaceAttributes <R L0.HasProperty
--> XIS.AttributeInfoItem
L0.InverseOf XIS.AttributeInfoItem.ownerElement
L0.HasDescription "An unordered set of attribute information items, one for each of the namespace declarations (specified or defaulted from the DTD) of this element. Declarations of the form xmlns=' ' and xmlns:name=' ', which undeclare the default namespace and prefixes respectively, count as namespace declarations. Prefix undeclaration was added in Namespaces in XML 1.1. By definition, all namespace attributes (including those named xmlns, whose [prefix] property has no value) have a namespace URI of http://www.w3.org/2000/xmlns/. If the element has no namespace declarations, this set has no members.":L0.String
>-- XIS.ElementInfoItem.isParentOf <R L0.IsComposedOf
--> XIS.ElementInfoItem
--> XIS.CharacterInfoItem
L0.InverseOf XIS.InfoItem.parent
L0.HasDescription "The document or element information item which contains this information item in its [children] property.":L0.String
>-- XIS.ElementInfoItem.elementType <R L0.IsRelatedTo : L0.FunctionalRelation
L0.HasDescription "An indication of the type found for this element during schema validation.":L0.String
```

On the other hand, Figure 7 presents actual element data (i.e. the content of imported REQ-IF-HEADER element) in Simantics.

It can be observed that XML elements map to instances of ElementInfoItems. However, they have elementType relation to actual simple or complex type definition of schema. This information is known according to PSVI provider of XML parser. This kind of design gives a lot of flexibility for element contents. Thus, different kind of XML content can be imported although all schemas are not available. In addition, usage of element and attribute declarations as relations and properties respectively has been shown.

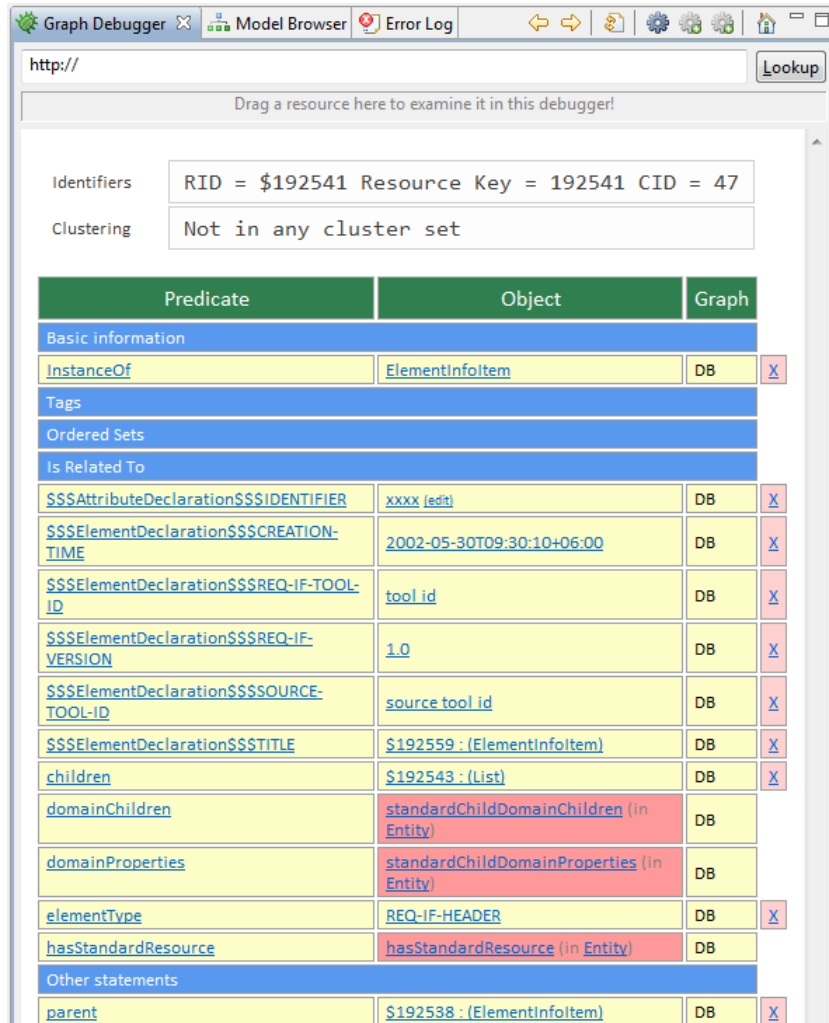
The Simantics XML data import functionality is based on implementation of SAX2 event handler (DefaultHandler). The class

```
org.simantics.xml.ui.browser.handler.MyContentHandler
```

implements this behaviour. Element character content and attribute values are mapped from XML data types into types of Simantics by the following way:

- anyType, string, NMTOKEN, NMTOKENS, IDREF, IDREFS, ID, ENTITY, ENTITIES, NCName, Name, language, token, normalizedString
 - L0.String
- anyURI, QName, gYear, gYearMonth, gMonth, gMonthDay, duration, dayTimeDuration, yearMonthDuration, dateTime, dateTimeStamp, date
 - L0.String
- integer, long, int, short, byte, nonNegativeInteger, positiveInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte, nonPositiveInteger, negativeInteger
 - L0.Long // also L0.Integer and L0.Byte would be available
- boolean
 - L0.Boolean
- decimal, double
 - L0.Double
- float

- L0.Float



Graph Debugger | Model Browser | Error Log

http://

Drag a resource here to examine it in this debugger!

Identifiers: RID = \$192541 Resource Key = 192541 CID = 47

Clustering: Not in any cluster set

Predicate	Object	Graph
Basic information		
InstanceOf	ElementInfoItem	DB <input type="button" value="X"/>
Tags		
Ordered Sets		
Is Related To		
\$\$\$AttributeDeclaration\$\$\$IDENTIFIER	xxxx <input type="button" value="edit"/>	DB <input type="button" value="X"/>
\$\$\$ElementDeclaration\$\$\$CREATION-TIME	2002-05-30T09:30:10+06:00	DB <input type="button" value="X"/>
\$\$\$ElementDeclaration\$\$\$REQ-IF-TOOL-ID	tool id	DB <input type="button" value="X"/>
\$\$\$ElementDeclaration\$\$\$REQ-IF-VERSION	1.0	DB <input type="button" value="X"/>
\$\$\$ElementDeclaration\$\$\$SOURCE-TOOL-ID	source tool id	DB <input type="button" value="X"/>
\$\$\$ElementDeclaration\$\$\$TITLE	\$192559 : (ElementInfoItem)	DB <input type="button" value="X"/>
children	\$192543 : (List)	DB <input type="button" value="X"/>
domainChildren	standardChildDomainChildren (in Entity)	DB
domainProperties	standardChildDomainProperties (in Entity)	DB
elementType	REQ-IF-HEADER	DB <input type="button" value="X"/>
hasStandardResource	hasStandardResource (in Entity)	DB
Other statements		
parent	\$192538 : (ElementInfoItem)	DB <input type="button" value="X"/>

Figure 7. An element of XML document.

Today, the mapping implementation is quite trivial but it can be made more sophisticated with the Simantics Databoard and its type system.

Because the design and implementation of graphical user interface to manipulate ReqIF requirements in Simantics was the out of scope of this research the later steps in the ReqIF data roundtrip are not included in this report. However, exporting manipulated ReqIF data from Simantics is not so challenge problem as the presented data import.

5. Conclusions

This report focused on the steps to support ReqIF data roundtrip and data validation in the Simantics environment. The report started by presenting a holistic user point of view of ReqIF data roundtrip. After that, the problem statement was generalized for the Simantics environment and alternative approaches to tackle the challenges were presented. In addition, the needed technologies for the selected approach were documented and rationales for the choices were presented. Finally, the demonstration was shown to concretize and visualize the results of this research. The work illustrated that deep knowledge of XML Schema and its abstract data model are necessary to survive with Simantics ReqIF data roundtrip implementation. Research questions related to full data roundtrip are still open and waiting for new features of Simantics.

References

- DOM 3. 2004. Document Object Model (DOM) Level 3 Core Specification. Version 1.0. W3C Recommendation 07 April 2004. <http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/>. Referenced 7 Oct 2014.
- Dongwon, L., Wesley W. C. 2000. Comparative Analysis of Six XML Schema Languages. Department of Computer Science, University of California, Los Angeles. <http://www.cobase.cs.ucla.edu/tech-docs/dongwon/ucla-200008.html>. Referenced 5 Nov 2014.
- Eclipse. 2015. The Eclipse Foundation. <http://www.eclipse.org/>. Referenced 8 Oct 2015.
- ISO/IEC 19757-3. 2006. Information technology — Document. Schema Definition Languages (DSDL) — Part 3: Rule-based validation — Schematron. First edition. 1 June 2006. <http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html>. Referenced 5 Nov 2014.
- JAXP 1.4. Trail: Java API for XML Processing (JAXP) in The Java™ Tutorials. Oracle. <http://docs.oracle.com/javase/tutorial/jaxp/>. Referenced 17th March 2014.
- Jelliffe, R. 2001. The Schematron. An XML Structure Validation Language using Patterns in Trees. Academia Sinica Computing Centre, Taipei. <http://xml.ascc.net/resource/schematron/schematron.html>. Referenced 5 Nov 2014.
- JSR 173. 2014. Streaming API for XML (StAX). Version 1.3. 4 Mar 2014. <https://jcp.org/en/jsr/detail?id=173>. Referenced 7 Oct 2014.
- JSR 206. 2013. Java™ API for XML Processing (JAXP). Version 1.6. <https://www.jcp.org/en/jsr/detail?id=206>. 4 Dec 2013. Referenced 7 Oct 2014.
- JSR 222. 2009. Java™ Architecture for XML Binding (JAXB) 2.2. Sun Microsystems, Inc. (Kawaguchi, K., Vajjala, S., Fialli, J.), Final Release. December 10, 2009. <https://jcp.org/en/jsr/detail?id=222>. Referenced 5th Oct 2015.
- Litani, E. 2004. XML Schema API. W3C Member Submission 9 Mar 2004. <http://www.w3.org/Submission/2004/SUBM-xmlschema-api-20040309/>. Rereferenced 27th Oct 2014.
- Murata, M., Dongwon, L., Murali, M. 2001. Taxonomy of XML Schema Languages Using Formal Language Theory. Presented at eXtreme Markup Language 2001. <http://www.cobase.cs.ucla.edu/tech-docs/dongwon/mura0619.pdf>. Referenced 5 Nov 2014.
- OSGi. 2015. The Dynamic Module System for Java™. OSGi Alliance. <http://www.osgi.org/Technology/WhatIsOSGi>. Referenced 8 Oct 2015.
- RELAX NG. 2001. RELAX NG Specification. Committee Specification 3 December 2001. The Organization for the Advancement of Structured Information Standards [OASIS]. <https://www.oasis-open.org/committees/relax-ng/spec-20011203.html> or https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=relax-ng. Referenced 5 Nov 2014.
- ReqIF. 2013. Requirements Interchange Format (ReqIF). Version 1.1. OMG Object Management Group. October 2013. <http://www.omg.org/spec/ReqIF/>. Referenced 6th Feb 2014
- SAX 2. 2002. Simple API for XML. Version 2.0.1. <http://sax.sourceforge.net/>. Referenced 7 Oct 2014.
- Simantics. 2014. THTH Association of Decentralized Information Management for Industry. Simantics Division. <https://www.simantics.org/simantics>. Referenced 7 Oct 2014.
- Simantics. 2015. http://dev.simantics.org/index.php/Main_Page. 25 June 2015. Referenced 7 Oct 2015.
- TrAX. 2006. Transformations API for XML. <http://xml.apache.org/xalan-j/trax.html>. Referenced 5th October 2015.
- Van der Vlist, E. 2001. Comparing XML Schema Languages. O'Reilly Media, Inc. 12 Dec 2001. <http://www.xml.com/pub/a/2001/12/12/schemacompare.html>. Referenced 5 Nov 2014.

- Woodstox. 2015. <http://wiki.fasterxml.com/WoodstoxHome>. Referenced 1 Sep 2015.
- Xerces2. 2010. Xerces2 Java Parser 2.11.0 Release. The Apache Software Foundation. <https://xerces.apache.org/xerces2-j/>. Referenced 5th October 2015.
- XML 1.0. 2008. Extensible Markup Language (XML) 1.0. Fifth Edition. W3C Recommendation 26 November 2008. <http://www.w3.org/TR/xml/>. Referenced 5 Nov 2014.
- XML 1.1. 2006. Extensible Markup Language (XML) 1.1. Second Edition. W3C. 16 August 2006. <http://www.w3.org/TR/2006/REC-xml11-20060816/>. Referenced 6th Feb 2014
- XML Catalogs. 2005. XML Catalogs OASIS Standard V1.1. Walsh, N. (ed). 7 Oct 2005. <https://www.oasis-open.org/committees/download.php/14809/xml-catalogs.html>. Rereferenced 27 Oct 2014.
- XML InfoSet. 2004. XML Information Set. Second Edition. W3C. 4 February 2004 <http://www.w3.org/TR/2004/REC-xml-infoset-20040204/>. Referenced 6th Feb 2014
- XML Namespaces. 2006. Namespaces in XML 1.1. W3C. 16 August 2006. <http://www.w3.org/TR/xml-names11/>. Referenced 6th Feb 2014
- XML Schema languages. 2014. Wikipedia. 17 Oct 2014. http://en.wikipedia.org/wiki/XML_schema_languages. Referenced 5 Nov 2014.
- XML Schema. 2004. XML Schema Part 0: Primer. W3C Recommendation 28 October 2004. <http://www.w3.org/TR/xmlschema-0/>. Referenced 6th Feb 2014.
- XMLBeans. 2012. Apache XMLBeans. Version 2.6.0. 14 Aug 2012. <http://xmlbeans.apache.org/>. Referenced 7 Oct 2014.
- XPath 3.0. 2014. XML Path Language (XPath) 3.0. W3C Recommendation 08 April 2014. <http://www.w3.org/TR/xpath-30/>. Rereferenced 7 Oct 2014.
- XQuery 3.0. 2014. An XML Query Language. W3C Recommendation 08 April 2014. <http://www.w3.org/TR/xquery-30/>. Rereferenced 7 Oct 2014.
- XSD 1.1. 2012a. W3C XML Schema Definition Language (XSD) 1.1 - Part 1: Structures. W3C Recommendation 5 April 2012. <http://www.w3.org/TR/xmlschema11-1/>. Rereferenced 7 Oct 2014.
- XSD 1.1. 2012b. W3C XML Schema Definition Language (XSD) 1.1 - Part 2: Datatypes. W3C Recommendation 5 April 2012. <http://www.w3.org/TR/xmlschema11-2/>. Referenced 17th March 2014.