



Masterarbeit zum Thema

Implementierung eines Webservices nach
ISO 29002-31
»Query for characteristic data«

Vorgelegt der
Fakultät für Mathematik und Informatik
Fernuniversität Hagen

Lehrgebiet Datenbanksysteme für neue Anwendungen

von
Stefan Sobek
Matrikelnummer 7736096

Eingereicht am
07. Februar 2014

Betreuer: Dr. Wolfgang Wilkes
Prof. Dr. Ralf Hartmut Güting

Zusammenfassung

Mit Beginn des 21. Jahrhunderts und dem Einzug des Internets in nahezu jeden Haushalt, jedes Geschäft und in die Industrie stehen Daten im absoluten Fokus. Schlagwörter wie »Big Data«, »Cloud Services« und »Mobile Devices« sind in aller Munde. Daten werden überall erfasst, gespeichert, ausgewertet und verarbeitet, seien es Multimedia-Daten wie Bild, Ton oder Film oder Personendaten und Produktdaten. Mittlerweile erfassen mobile Endgeräte wie z.B. ein Smartphone, aber auch Kraftfahrzeuge oder Internet-Webseiten laufend Daten. Bei Fahrzeugen ist dies beispielsweise die Position, Geschwindigkeit oder über Sensoren Werte wie Temperatur oder Luftdruck. Internetseiten sammeln Zugriffsdaten und analysieren das Verhalten der Nutzer. Solche Daten werden gespeichert und gegebenenfalls weiterverarbeitet. Sie fallen in schier unfassbaren Mengen an.

Oft spielt in diesen gesammelten Massendaten die Qualität der Daten eine untergeordnete Rolle, wobei in diesem Kontext mit Qualität eine möglichst präzise konzeptuelle Beschreibung der Daten gemeint ist. Solche Massendaten werden oft erst einmal gesammelt, um dann später aufwändig ausgewertet zu werden. Betrachtet man beispielsweise Stammdaten in der Industrie, findet man z.B. Daten zu Kunden, Lieferanten, Produkten, Materialien, Wirtschaftsgütern oder Angestellten. Die Qualität dieser Daten hat in der Industrie eine deutlich höhere Gewichtung. Der Zugriff und die Weiterverarbeitung muss schnell erfolgen. Die Unternehmen katalogisieren und beschreiben ihre Daten, sodass diese innerhalb und außerhalb der Organisation zwischen einzelnen Systemen ausgetauscht werden können und definiert ist, was die einzelnen Datensätze bedeuten. Nehmen wir beispielsweise einen Hersteller von Sechskantschrauben zur Befestigung von Rädern an Fahrzeugen. Die Information über diese herzustellende Schraube muss definiert und in der Produktion bekannt sein. Der Verkauf benötigt allerdings alle diese Informationen für den Vertrieb des Produktes. Ferner möchte der Kunde, der diese Schraube für die Produktion seines Fahrzeugs benötigt, selbstverständlich auch diese Information, zum einen für den Einkauf, zum anderen für die Planung und Fertigung, denn es muss bekannt sein aus welchen Einzelteilen (Stückliste) ein Produkt besteht. Diese Daten müssen folglich zur Verfügung gestellt werden.

Es ist ersichtlich, dass Stammdaten in den verschiedenen Bereichen benötigt werden. Zu nennen seien beispielsweise die Lieferketten, das Design und die Herstellungsprozesse als auch im Produktlebenszyklusmanagement.

Wenn die Beschreibung der Daten nicht vollständig ist, wenn gleichsam Informationen zu Stammdaten an einem Punkt der Lieferkette oder im Produktlebenszyklusmanagement fehlen oder ungenau sind, so führt das zu Problemen und hohem Kostenaufwand. Diese Daten in Konzepte zu beschreiben, diese Konzepte verfügbar zu

machen und diese Daten automatisiert auszutauschen ist das Ziel. Der automatisierte Austausch von Produktdaten ist in der Praxis übliches Vorgehen und durch Prozesse abgebildet. Damit diese Daten automatisiert vom Sender zum Empfänger gelangen, müssen sie übertragen werden. Dazu muss ein genaues Modell der Daten definiert sein, z.B. als XML-Repräsentation. Herkömmlicherweise ist das Modell als Schema definiert, sodass der Sender und Empfänger den Aufbau der Daten kennen. Allerdings stellt sich hier das Problem, dass die Schnittstellen und Schemata oft starr und unflexibel sind. Anpassungen daran sind häufig mit hohem Einsatz und hohen Kosten verbunden. Jedoch sind Flexibilität und Änderbarkeit sehr wichtige Eigenschaften, um schnell und effizient auf Änderungen an den Anforderungen reagieren zu können.

In den Abschlussarbeiten des Fachbereiches geht es um Problemstellungen rund um die PLIB (Parts Library), die sich mit der Beschreibung eines Datenmodells für Dictionaries und Bibliotheken befasst. Im Rahmen der Untersuchungen werden als mögliche Lösungen ISO-Standards zu diesen Themen analysiert und implementiert.

Diese Arbeit handelt von der Problemstellung der automatisierten Austauschbarkeit von charakteristischen Produktdaten, das sind Eigenschaften, die ein Produkt möglichst präzise und eindeutig beschreiben, um einen bestimmten Zweck zu erfüllen. Dabei werden mögliche Lösungsoptionen für diese unflexiblen Austauschschnittstellen betrachtet. Eine Lösungsoption sind flexible Abfrage- und Antwortschnittstellen. Hier kommt beispielsweise der ISO Standard »ISO 29002-31 - Query for characteristic data« als Beschreibung einer flexiblen Abfrage- und Antwortschnittstelle in Frage. Nach einer Analyse des ISO-Standards wird darauf aufbauend eine Implementierung eines RESTful Webservices vorgenommen. Eine Anfrage der Abfrage- und Antwortschnittstelle gemäß »ISO 29002-31 - Query for characteristic data« referenziert mittels eindeutigem Identifier (IRDI) Ontologien. Das Ergebnis einer Abfrage ist eine Antwort, welche die Eigenschaftsdaten der angefragten Produktdaten beinhaltet. Dazu ist ebenfalls eine Ontologie nötig, welche der Standard »ISO 29002-10 - Characteristic data exchange format« beschreibt. Hierin wird das Datenformat der zurückgelieferten Daten spezifiziert. Die Datenquelle für den Webservice ist die PLIB-Datenbank, eine Implementierung einer Produktdatenbank nach ISO 13584-42, welche die referenzierten Ontologiekonzepte liefert.

Aufbauend auf die Implementierung der Anfrageverarbeitung lässt sich anschließend sehr einfach ein weiterer Webservice basierend auf SOAP implementieren. Da bereits ein Modell für die gewählte Programmiersprache im Rahmen der RESTful Webservices Implementierung generiert wird, kann eben diese benutzt werden. Lediglich eine weitere Implementierung der Schnittstellentechnologie basierend auf einem Java Framework ist nötig.

Aus dem Zusammenspiel mehrerer Standards lässt sich eine Implementierung sinnvoller Anwendungsfälle erstellen. Ein sinnvoller Anwendungsfall wäre beispielsweise die

Anfrage nach allen Eigenschaften eines Produkts, welches durch einen weltweit eindeutigen Identifier identifiziert ist. Die Anfrage wird per XML nach ISO 29002-31 formuliert und an einen Webservice gesendet und verarbeitet. Die Antwort wird als XML nach ISO 29002-10 formuliert und an den Konsumenten zurückgesendet. Komplexere Fälle schränken die Anfrage nach bestimmten Wertebereichen einer Eigenschaft ein oder selektieren nur bestimmte Eigenschaften, welche dann als Antwort zurückgesendet werden sollen. Dies entspricht etwa einer flexiblen Abfrage- und Antwortschnittstelle, wie sie beispielsweise von SQL bekannt ist.

Während der Analyse und Implementierung ergeben sich diverse Herausforderungen und Problemstellungen, welche es zu bewältigen gilt, seien es Heterogenität in der Datenrepräsentation der aufzurufenden Datenbankprozeduren, die Datenquelle, oder technische Problemstellungen wie z.B. bei der Generierung der Modellklassen zur Weiterverarbeitung der Anfrage im System oder fehlende Unterstützung von benutzten Typen aus der Oracle Datenbank in den eingesetzten Programmiersprachen und Frameworks.

Abstract

With the beginning of the 21st century, and the global success of the internet in all households and businesses, collecting and processing data is more and more in the daily focus. Buzzwords like »Big Data«, »Cloud Services« and »Mobile Devices« are broadcast through the internet and in the media. Data will be collected in nearly every area like Multimedia-data as pictures, audio or video data, or other data types like personal and product data. This data could for example be collected by Mobile devices as well as cars or trucks or through websites in the internet. Cars and trucks collect data in type of position, current speed or via sensors the temperature or barometric pressure. Websites collect and analyse data regarding the user's behavior on the website. Such data will then be saved and further processing will take place as the amount of data is enormous.

When collecting such volumes of data, the quality of the data is not always the most important factor. In this context with the term »quality« the precise conceptual description of data is meant. The data will be collected and later be analysed through a time-consuming process. Considering master data in the industry, manufacturing and trade there can be observed data entities like customer, supplier, material, economic goods or simply personal data regarding employees. The quality of the data has a pretty high importance here. Companies create catalogues and descriptions of their data so that the information can be exchanged between several systems. It is very important to know precisely what the datasets mean in their context. A manufacturer for hexagon bolts, which are used to mount tyres on cars, needs exact information, description and definition of this particular product for its production division. Furthermore Sales needs the information to sell the product to interested customers and provide them with all information about the product. Last but not least, the customer needs precise information about the product, in his design and production department, for the parts list which defines the needed parts for his concrete product. Master data is needed in nearly every area, in supply chain, design and production as well as Product-Lifecycle-Management.

If the description of the data is not complete or not precise or information is missing at some point in the supply chain or in Product-Lifecycle-Management, this can lead to problems and higher costs. The main goal is to describe data in concepts, make those concepts available and create standards which allow to automate the data flow. The automated exchange of product data is common in practice and this is realised through business processes. A precise model of the product data must be defined to transfer the product data from sender to receiver and this is often realised with a XML-representation of the data. The model is defined via schema, so that the sender and the receiver understand the meaning of the data and their properties.

The problem with automated product data transfer is that interfaces and schemas are fixed and unflexible. Modifications and updates on interfaces and schemas are time consuming and costly, however, flexibility is rather important nowadays as we must react fast and efficiently on changes to specifications.

All current thesis work in the area of studies around PLIB topics, which describes a data model for dictionaries and libraries, consider the above mentioned problems. To support solving the problems several ISO-standards regarding these topics will be considered and implemented.

This thesis considers the problem regarding automated transfer of characteristic product data based on the PLIB data model. The ISO-standard »ISO 29002-31 - Query for characteristic data«, which is a description of a flexible query-response interface, will be implemented as a Webservice due to this purpose.

The ISO-standard »ISO 29002-31 - Query for characteristic data« is the main query-interface, which references ontologies via unique identifier (IRDI). The response contains the data of the properties of the requested product data. An ontology is needed for that as well which will be described in »ISO 29002-10 - Characteristic data exchange format«. It describes the data format of the data responded. The data source for the Webservice is the PLIB-database, an implementation of a product database according to ISO 13584-42.

Thus meaningful use cases are a conjunction of several Standards. A simple use case is, for example, the request for all properties of a product which is identified by a unique identifier. The request will be sent via XML according to ISO 29002-31 and to a webserver and then processed. The response will be sent according to ISO 29002-10 as an XML file to the consumer. More complex cases will restrict the request to a specific value range of the requested properties or will select specific properties only. These will be then sent back as response. This query-response interface correlates pretty much to a flexible query-response interface like it is known by SQL.

During the analysis and implementation several challenges and problems can occur which must be handled. For example, heterogeneity in the data representation of the called database procedures which will deliver the requested data or technical problems with the generation of the model classes for further processing in the system. Another problem is that a specific data type used in the procedures of the Oracle database has no support in the used programming language and framework.

Inhaltsverzeichnis

Zusammenfassung	III
Abstract	VII
Abbildungsverzeichnis	XIII
Tabellenverzeichnis	XIV
Listingverzeichnis	XVI
Glossar	XIX
1. Einleitung	1
2. Aufgabenbeschreibung	3
2.1. Problemstellung	3
2.2. Zielsetzung	4
2.3. Kontext und Abgrenzung	6
2.3.1. Gesamtkontext der PLIB Abschlussarbeiten	6
2.3.2. Abgrenzung	7
2.3.3. Vorgaben	7
2.3.4. Datenflüsse	8
3. Analyse und Definition der Anforderungen	11
3.1. Analyse ISO 29002-31 - Exchange of characteristic data	11
3.2. Analyse ISO 22745-30 - Identification Guide	12
3.3. Anwendungsfälle	13
3.3.1. Was ist ein Use Case?	13
3.3.2. Akteure	14
3.3.3. Use Case Beschreibungen	14
3.4. Zusammenfassung des Kapitels	16
4. System- und Softwareentwurf	19
4.1. Auswahlprozess	19
4.1.1. Webservice	19
4.1.2. Plattform	21
4.1.3. Bibliotheken und Frameworks	21
4.1.4. Programmiersprache	22
4.2. Softwaredesign und Architektur	22
4.2.1. Bausteinsicht	22
4.2.2. Kontextabgrenzung - Systemüberblick und angrenzende Systeme .	22
4.2.3. Level 1 - Plib characteristic query	23

5. Implementierung	25
5.1. Configuration Management und Setup	25
5.1.1. Versionsverwaltung	25
5.1.2. Versionsverwaltung mit GIT	25
5.1.3. Build Management mit Maven	26
5.1.4. Apache Tomcat	26
5.1.5. Tests	27
5.2. Webservice	28
5.2.1. Servlet Konfiguration in web.xml	28
5.2.2. Webservice Klasse	29
5.3. Query-Verarbeitung	30
5.3.1. Modell-Generierung	31
5.3.2. Einbinden in Buildprozess mit Maven	33
5.4. Transformation und Abfrage der PLIB-Prozeduren	35
5.4.1. Problemstellung - Externer Identifier	38
5.4.2. Problemstellung - Daten der Teile-Eigenschaften	39
5.4.3. Problemstellung - Aufruf der Prozeduren mit Java	41
5.4.4. Problemstellung - RECORD-Types als Argumente werden nicht von Java-JDBC unterstützt	45
5.4.5. Problemstellung - Fehlender Fremdschlüsselidentifier in Prozessrückgabedaten	47
5.5. Fehlerbehandlung	50
5.6. SOAP Webservice	52
5.7. Umfang und Abgrenzung der Implementierung	52
5.8. Zusammenfassung des Kapitels	53
Schlussfolgerung	55
Literaturverzeichnis	57
Anhang	60
A. Analyse ISO 29002-31 - Exchange of characteristic data	61
A.1. XML Datencontaineranalyse ISO 29002-31	61
A.1.1. query_context	61
A.1.2. query	62
A.1.3. characteristic_data_query_expression (parametric_query)	62
A.1.4. Query Beispiele	64
A.2. Analyse ISO 22745-30 - Identification Guide	65
B. Anwendungsfälle	67
B.0.1. Charakteristische Daten eines Produkts validieren	67
B.0.2. Charakteristische Daten mittels Suchausdruck abfragen	68

C. Installationsanleitung Apache Tomcat 7	71
C.1. Installation auf Mac OS 10.8	71
C.1.1. Prüfen der Java Version	71
C.1.2. Tomcat herunterladen und entpacken	71
C.1.3. Tomcat starten	72
C.1.4. Tomcat stoppen	72
C.2. Installation unter Windows	72
C.3. Tomcat mittels Maven starten	72
D. Automatische Entwicklertests	73
D.1. Unit Test	73
D.2. Integrationstest	76
E. Testszenarios und Testtools	81
E.1. Simple query	81
E.1.1. Vorhandene Teile anhand IRDI abfragen	81
E.1.2. Ungültige IRDI angegeben	81
E.1.3. Vorhandene Teile anhand IRDI abfragen mit Projektion	82
E.1.4. Vorhandene Teile mit bekannten Werten validieren	82
E.2. Parametric query	83
E.2.1. Abfrage mit Werteeinschränkung auf eine Eigenschaft eines Items	83
E.3. Testtools	83
E.3.1. CURL	84
E.3.2. Advanced REST Client	84
F. PLIB Datenbanktabellen - Ausschnitt	85
G. SOAP Webservice	87
G.1. Erstellung einer Webservice-Klasse	87
G.2. Erstellung eines Webservice Servers	88
G.3. SOAP Beispielrequest und -response	89
G.4. Fazit	91
H. Architektur - Bausteinsichten	93
H.1. Level 2 - Whiteboxansicht - Komponente XMLData	93
H.2. Level 2 - Whiteboxansicht - Komponente Processor	94
H.3. Level 3 - Whiteboxansicht - Komponente Handler	94
H.4. Level 3 - Whiteboxansicht - Komponente Analyser	95
I. Schema Dateien	97
I.1. query.xsd	97
I.2. catalogue.xsd	101
I.3. basic.xsd	102
I.4. identifier.xsd	104

I.5. value.xsd	105
Index	111

Abbildungsverzeichnis

2.1. Gesamtkontext PLIB Abschlussarbeiten	6
2.2. Lieferketten	7
2.3. Datenflüsse	9
3.1. Mögliche Abfragen anhand der Kombination der Daten in XML Query . .	12
3.2. Use Case Übersicht	14
4.1. Bausteinsicht - Kontextabgrenzung	23
4.2. Bausteinsicht - Level 1	24
5.1. Sequenzdiagramm Simple Query	40
A.1. UML-Diagramm Query Main	61
A.2. UML-Diagramm Query Expression	63
E.1. Advanced REST Client Test des Webservices	84
F.1. PLIB Datenbanktabellen - Ausschnitt	86
H.1. Bausteinsicht - Level 2 - Komponente XMLData	93
H.2. Bausteinsicht - Level 2 - Komponente Processor	94
H.3. Bausteinsicht - Level 3 - Komponente Handler	95
H.4. Bausteinsicht - Level 3 - Komponente Analyser	95

Tabellenverzeichnis

2.1. Erläuterung der Gesamtkontextabbildung	6
5.1. Erläuterung des Abfrageablaufes der Applikation	40

Listings

3.1. Query Beispiel - alle Daten abfragen	16
3.2. Query Beispiel - Daten abfragen mit Propertyeinschränkung (Projektion)	16
5.1. Jersey Servlet Konfiguration in web.xml	28
5.2. Jersey Servlet Mappingkonfiguration in web.xml	29
5.3. Jersey Webservice Klasse	29
5.4. Jersey Methode	29
5.5. query.xsd Namespace Definitionen	32
5.6. Binding File binding.xjc	32
5.7. Build Helper Maven Plugin	34
5.8. JAXB Maven Plugin	34
5.9. SQL Query - Externe IDs abfragen	39
5.10. Spring Data Oracle - Klasse zum Aufruf der Hilfsprozeduren	42
5.11. Spring Data Oracle - Testklasse zum Aufruf der Hilfsprozeduren	43
5.12. Spring Data Oracle - Aufruf der Prozeduren mit Rückgabetabellen	44
5.13. SELECT-Abfrage für String-Properties aus den Prozeduren	46
5.14. PROP_STRING_OBJ Typen	47
5.15. PROP_STRING_OBJ Typanpassung	48
5.16. GET_OBJ_STRING Anpassung auf Property ID	48
5.17. Fehlerbehandlung - Error Schemadatei	50
5.18. Fehlerbehandlung - Catalogue Type erweitern	51
5.19. Fehlerbehandlung - Beispielantwort mit Validierungsfehler	51
B.1. Query Beispiel - Daten validieren	68
B.2. Query Beispiel - Daten mit Suchausdruck abfragen	69
C.1. Tomcat 7 Maven Plugin	72
D.1. Beispiel eines Unit Tests	73
D.2. Abstrakte Unit Testklasse	74
D.3. Beispiel eines Integrationstests	76
E.1. CURL Test des REST Webservices	84
G.1. SOAP Webservice Klasse zum Senden eines Queries	87
G.2. SOAP Webservice Server	88
G.3. Beispielanfrage SOAP Webservice query	89
G.4. Beispielantwort SOAP Webservice response	89
G.5. WSDL des Query Services mit SOAP	90
I.1. query.xsd	97
I.2. catalogue.xsd	101

I.3.	basic.xsd	102
I.4.	identifier.xsd	104
I.5.	value.xsd	105

Glossar

Abfrage- und Antwort Schnittstelle Auch Query-Response-Schnittstelle. Eine flexible Schnittstelle über die vom Klienten eine Anfragenachricht an einen Server gesendet wird, welcher eine Antwortnachricht zurückliefert. Die Anfragen sind flexibel dadurch, dass Werte- und Attributeinschränkungen gemacht werden können. IV, V, VIII, 1, 4, 5, 16, 55

Annotation Metainformation in der Java Programmierung. Wird zur Reflektion des Codes genutzt. 30

Apache Tomcat Web Container zum Ausführen und Ausliefern von Webseiten basierend auf Java. 21, 26, 28

Applikationskontext Auch application context. Die Ausführungsumgebung einer Software Applikation innerhalb eines Applikationsservers. Wird an eine URL gekoppelt, unter der die Applikation erreichbar ist. 29

Dictionary Ein Verzeichnis, welches Daten zu Konzepten enthält. Diese Konzepte werden mittels Eigenschaften präzise beschrieben, sodass im besten Falle die Beschreibung eindeutig ist. IV, 63

ECCMA Electronic Commerce Code Management Association ist eine internationale nicht gewinnorientierte Organisation, mit dem Ziel bessere Qualitätsstandards für elektronische Daten zu erforschen, zu entwickeln und zu verbreiten. 31

Excel Microsoft Excel ist ein bekanntes Tabellenkalkulationsprogramm der Firma Microsoft.. 7

HTTP Hyper Transfer Protocol. Protokoll zur Übertragung von Daten über ein Netzwerk. Wird in der Anwendungsschicht angesiedelt und ist das hauptsächlich eingesetzte Protokoll, um Webseiten im Internet (World Wide Web) zu übertragen. 19, 20, 30, 50

HTTP-Methode Eine HTTP-Anfrage an einen Server, gemäß des Standards, z.B. eine GET-, POST-, PUT- oder DELETE Anfrage. Es werden dadurch Ressourcen auf einem Server abgefragt, geändert oder gelöscht. 30

Identification Guide Ein Identification Guide beschreibt eine Menge von Regeln für die Beschreibung von Teilen (Items), welche zu einer bestimmten Klasse gehören. Hierbei werden die Eigenschaften und Klassendefinition zu Konzepten eines Dictionaries verlinkt. 7, 12, 17, 63

Integrationstest Ein Integrationstest ist ein Test einer Komponente integriert in die System- oder Komponentenlandschaft. Es wird die Komponente in Zusammenarbeit mit anderen Komponenten im Rahmen von Szenarios getestet. Z.B. Test einer grafischen Benutzeroberfläche mit Speicherung von Eingabedaten in der Datenbank. 27

Interoperabilität Die Fähigkeit homogener Systeme Informationen effizient auszutauschen. 3

IRDI International Registration Data Identifier, global eindeutiger Identifizierer für ein Objekt oder Konzept. IV, V, VIII, 4, 15, 21, 38, 39, 43, 60, 63

JAXB Java Architecture for XML Binding, Programmierschnittstelle, die Daten aus XML-Schemata an Java-Klassen bindet. 31, 32

Jersey Framework zur Erstellung von RESTful Webservices in Java. 21, 28

Marshalling Beschreibt das Umwandeln von einem programmatischen Modell, z.B. einer Java Klasse, in XML Daten. 71

Maven Ein Build-Management-Tool der Apache-Foundation. Wurde mit Java entwickelt und ermöglicht, Java-Programme standardisiert zu erstellen und zu verwalten. 33, 34

MIME-Type Internet Media Type, wird auch Content-Type genannt. Klassifiziert die Daten im Kopfbereich einer Nachricht. Hiermit wird dem Empfänger mitgeteilt, welche Art der Daten gesendet werden. Das können beispielsweise reine Textdaten, Bild-, XML- oder Videodaten sein. 30

Namespace XML-Namensräume werden benutzt, um mehrere verschiedene Vokabulare in einer XML-Datei zu unterscheiden. 31, 32

Ontologie Der Begriff Ontologie bezeichnet das Studium der Kategorisierung von Dingen, die existieren oder in einigen Interessensgebieten bestehen können. Das Ergebnis einer solchen Studie, eine Ontologie, ist ein Katalog von Typen von Dingen D, von denen man annimmt, dass sie aus der Sicht einer Person, die eine Sprache L benutzt, um über diesen Interessensbereich D zu sprechen, existieren. IV, VIII, 4, 55

Oracle Oracle Corporation, einer der weltweit größten Softwarehersteller. Bekanntestes Produkt ist das Datenbankmanagementsystem Oracle Database. Oracle übernahm im Jahre 2009 Sun Microsystems und somit die Entwicklung der Java Programmiersprache. 39, 53

PLIB Parts Library gemäß ISO 13584-42, beschreibt ein Datenmodell für Dictionaries und Bibliotheken. IV, VIII, 5, 28, 35, 52, 53, 55, 56

pom.xml Project Object Model - Konfigurationsdatei eines Maven Projektes. 34, 70

POST Eine HTTP Anfrage Methode, POST wird für das Übertragen von theoretisch unbegrenzter Menge an Daten an einen Server eingesetzt. 30

Produktlebenszyklusmanagement Auch Product-Lifecycle-Management, ist ein Konzept zur nahtlosen Integration sämtlicher Informationen, die im Verlauf des Lebenszyklus eines Produktes anfallen. III, VII

REST Representational State Transfer. Ein Architekturmodell basierend auf dem HTTP-Protokoll. Häufig genannt in Verwendung mit RESTful Webservices. IV, 6, 19–21, 23, 28, 50, 52, 53, 55, 82, 85

Servlet Servlets sind Java Klassen, welche nur innerhalb eines Webservers laufen. Diese nehmen Anfragen von Clients entgegen und beantworten sie. 28, 29

SOAP Simple Object Access Protocol. Protokoll für Nachrichten, diese werden zwischen Webservice-Konsument und Webservice-Anbieter ausgetauscht. IV, 6, 19–21, 23, 50, 52, 55

Spring Ein nicht invasives Open Source Applikationsframework mit dem Ziel, die Softwareentwicklung zu vereinfachen. 22

Stakeholder Stakeholder sind Projektbeteiligte, gleichsam Personen oder Institutionen und Dokumente, die in irgendeiner Weise vom Betrieb des Systems betroffen sind. 13

Teil Mit Teil, (englisch) Item oder auch Instanz genannt, ist eine Instanz eines konkreten Konzeptes der Teiledatenbank gemeint. Dieses Teil referenziert das Konzept und enthält konkrete Datensätze samt Wert, z.B. könnte es den Wert einer Eigenschaft eines Konzeptes enthalten. 15, 16, 35, 39, 47, 60, 62, 65, 66, 81

Unit Test Ein Unit Test ist ein Test der Funktionalität einer Komponente ohne Einbindung der Abhängigkeiten von anderen Komponenten. Dieser Test ist somit ein Whiteboxtest, da die Komponente völlig autark auf Funktionalität getestet wird. 27

Unmarshalling Beschreibt Umwandeln von XML Daten in programmatisches Modell, z.B. in Java Klassen. 30, 71

URI Uniform Resource Identifier, identifiziert eine abstrakte oder physische Ressource wie z.B. eine Webseite, ein E-Mail Empfänger oder eine Person. 29

URL Uniform Resource Location, identifiziert und lokalisiert eine Ressource, wie beispielsweise eine Webseite oder ein Bild über die verwendeten Netzwerkprotokolle. 29, 86

Use Case Use Case oder auch Anwendungsfall, beschreibt meist in Textform das Verhalten eines bestimmten Systems in Interaktion mit einem Benutzer in verschiedenen Szenarien. 13, 14, 17

Versionskontrollsystem Wird zur Versionsverwaltung eingesetzt. Erfasst Änderungen an Dokumenten oder Dateien, um diese später wiederherstellen zu können. Unterstützt häufig das gemeinsame Bearbeiten von Dateien und Dokumenten. 25, 26, 34

Webservice Oft auch als Web Service oder web service (Englisch) geschrieben. Stellt einen Web Dienst dar. IV, V, VIII, 6, 8, 14, 19–23, 28–30, 50, 52–55, 60, 85, 90

WSDL Web Service Description Language. W3C-Standard für die Beschreibung des Services und der Daten, die zwischen Konsument und Anbieter ausgetauscht werden. 19–21

1. Einleitung

Die Problemstellung dieser Arbeit sowie die Zielsetzung ist in Kapitel 2 beschrieben. Um aufzuzeigen, in welchem Kontext sich diese Arbeit befindet, sowie zum Zwecke der Abgrenzung der Aufgabe, wird eine Übersicht aller PLIB Abschlussarbeiten gegeben. Das Kapitel geht anschließend auf die technischen und fachlichen Vorgaben ein und gibt eine technische Beschreibung der Anforderungen.

Die zur Lösung der Problemstellung unterstützenden ISO Standards ISO 29002-31 und ISO 22745-30 werden in Kapitel 3 auf die Fragestellung hin analysiert, ob diese für die Implementierung einer flexiblen Abfrage- und Antwortsschnittstelle ausreichend sind. Das Ergebnis dieses Kapitels sind Anwendungsfälle mit entsprechenden Query-Beispielen basierend auf den Erkenntnissen der Analyse.

Im nächsten Schritt geht es darum, das System zu entwerfen. Die einzelnen Schritte des System- und Softwareentwurfes werden in Kapitel 4 erläutert. Dazu gehört eine Beschreibung des Auswahlprozesses der notwendigen Programmiersprache, der technischen Plattform, Frameworks und Architekturmuster. Ferner wird in diesem Kapitel der Architekturentwurf vorgestellt und erläutert. Dabei werden die Komponenten des Systems in Diagrammen dargestellt und weiter verfeinert, sodass eine Übersicht über das Gesamtsystem bis hin zu einzelnen fachlichen Verantwortlichkeiten gegeben wird.

Das Kapitel 5 beschreibt die entwickelte Lösung und geht auf wichtige Details der Implementierung ein. Ein besonderes Augenmerk richtet sich auf die Problemstellungen, welche sich vor und während der Implementierung ergeben. Die besonderen Herausforderungen, wie Integration und der Nutzen der vorhandenen Prozedurschnittstelle, welche vorgegeben wurde, bekommen eine genauere Betrachtung, da dies einer der Hauptaspekte dieser Arbeit darstellt.

Es folgt das Fazit im Schlussteil. Dieses Kapitel greift das Ergebnis auf und zeigt, welche weitere Möglichkeiten und Untersuchungen sich aus den Erkenntnissen ergeben. Es wird der Gesamtkontext der Arbeiten im Fachbereich aufgegriffen und ein Gesamtbild erstellt, um die einzelnen Teile und Arbeiten zusammenfügen zu können.

2. Aufgabenbeschreibung

Hast du etwas Schwieriges zu erledigen, beauftrage damit einen Faulen. Er wird einen leichten Weg finden, es auszuführen. . .

(Sokrates)

2.1. Problemstellung

Der automatisierte Austausch von Produktdaten spielt in der Geschäftswelt eine große Rolle. Es fallen in der Industrie, der Wissenschaft sowie in der Gesellschaft und Verwaltung große Datenmengen an, wie z.B. die Daten von Produkten, Bestellungen, Rechnungen, Lieferungen oder in der Verwaltung von Personen. Diese Datenmengen müssen nicht nur gespeichert sondern auch bereitgestellt respektive ausgetauscht werden. Der elektronische Datenaustausch erfolgt in der Regel nach einem festem Schema. Hierbei ist das Austauschformat sowie die Schnittstelle über die diese Daten ausgetauscht werden dem Anfragenden als auch dem Anbieter bekannt. Man »einigt« sich auf definierte Schemata. Dafür nutzt man definierte Standards, um Interoperabilität¹ zu gewährleisten. Zwei Applikationen sind interoperabel, wenn sie die terminologische Semantik in ihren korrespondierenden Konzepten gemeinsam nutzen und teilen (vgl. Hem09, S. 23f). Das bedeutet, dass zwei Parteien, welche die Anforderungen eines Standards bezüglich Schnittstellen erfüllen und entsprechend ihre zu übermittelnden Daten gemäß Schema formen, eben jene Daten miteinander austauschen können.

Basierend auf diese Standards werden Geschäftsprozesse definiert, um beispielsweise den Austausch der Daten und somit den Zugriff eines anderen ggfs. entfernten Systems auf diese Daten zu ermöglichen. Diese wohldefinierten Prozesse der Geschäftswelt sind in der Regel sehr unflexibel. Dadurch, dass das Modell der Daten sowie die Schnittstelle »fest verdrahtet« sind, bedeuten Änderungen in den Anforderungen bezogen auf Semantik, Struktur und Inhalte der Daten meistens eine Änderung des Modells, gleichsam der auszutauschenden Daten mit ihren Eigenschaften, als auch eine Anpassung der Schnittstelle. Diese Anpassungen sind in der Regel mit hohen Kosten verbunden, denn nicht nur anbieterseitig muss eine Anpassung erfolgen, der Konsument muss häufig über die Änderung informiert sein und seinerseits Anpassungen vornehmen. Ferner stellt sich das Problem, dass für komplexere Produktdaten auch komplexe autarke Schnittstellen geschaffen werden müssen, um diese Komplexität zuerst abilden und schließlich übertragen zu können. Folglich ist es wichtig, möglichst flexibel zu sein, gleichzeitig aber auch abstrakte Modelle, welche eine Trennung von Klassenkonzepten und Daten ermöglichen, zu unterstützen.

Der Anfrage- und Antwortprozess muss automatisierbar respektive maschinenlesbar sein, gleichzeitig fehlerfrei und flexibel (vgl. Uit12).

¹Die Fähigkeit homogener Systeme Informationen effizient auszutauschen.

2. Aufgabenbeschreibung

Um das Problem zu lösen, bieten sich flexible Mechanismen einer Abfrage- und Antwortsschnittstelle an. Typischerweise ist solch eine Abfrage- und Antwortsschnittstelle anbieterseitig im Datenbanksystem implementiert. Darauf aufbauend werden für diese Schnittstelle Methoden angeboten, welche diese Abfragen ermöglichen und passend zu diesen Abfragen entsprechende Antworten mit den gewünschten Daten erzeugen. Diese herkömmlichen Datenaustauschmechanismen eignen sich somit für eine konkrete Schnittstelle, definieren ein Schema zum Datenaustausch und sind eng an das Domänenmodell gekoppelt. Die flexiblen Abfragemechanismen sind anbieterseitig nahe der Datenquelle implementiert, zumeist bereits implizit vorhanden, beispielsweise durch ein Datenbanksystem mit SQL²-Abfrageschnittstelle.

Wie bereits erwähnt, baut herkömmlicherweise auf eine SQL-Schnittstelle eine Ebene mit Abfragemethoden auf, um den Zugriff für Business Funktionalität einfacher zu ermöglichen. Führt man eine weitere zusätzliche Ebene ein, die mittels eines Schemas eine flexible Abfrage- und Antwortsschnittstelle darstellt, so ermöglicht diese einen flexiblen und standardisierten Datenaustausch, wobei die tatsächlichen Werte der Daten vom eigentlichen Domänenmodell abgekoppelt sind. Diese lose Kopplung ist der große Vorteil dieser Lösung und ermöglicht eine große Anpassungsflexibilität. Konkret werden Daten über die Eigenschaften eines Produktes übermittelt, allerdings liegt der Nachricht keine Semantik bei. In der Nachricht wird mittels eines eindeutigen Identifiers auf eine Ontologie verwiesen. Der Anfragende muss folglich eine weitere Ontologie-Quelle abfragen, um Metainformationen über die Bedeutung des erhaltenen Wertes zu erlangen. Der Einsatz einer Ontologie als Anfrage- und Antwortmodell vereinfacht die Suche und das Browsing von Informationen (vgl. Hem09, S. 24f).

Mit diesem Problem haben sich bereits mehrere Interessensgruppen befasst, wie z.B. eCl@ss (vgl. Uit12). Es gibt einige Standards welche bei der Lösung unterstützen. Zu nennen ist ISO Standard 22745-35 oder ISO Standard 29002-31. Diese beschreiben eine solche Abfrage- und Antwortsschnittstelle auf Ontologiebasis. Die Standards zeigen den flexiblen automatisierten Austausch von charakteristischen Produktdaten auf. Diese Daten (meist Master Data) werden abgekoppelt von der Semantik abgefragt. Die Semantik wird mittels Klassenkonzepten beschrieben. Auf diese Klassenkonzepte wird wie bereits vorab beschrieben mittels eindeutigem Identifier referenziert und die konkreten Daten als Schlüssel-Wertepaare hinterlegt.

2.2. Zielsetzung

Ziel der Arbeit ist es eine flexible Abfrage- und Antwortsschnittstelle (Query-Schnittstelle) zu implementieren. Als Datenbasis mit notwendigen Katalog-, Klassenkonzept- und Metadaten soll die bereits vorhandene PLIB³ des Fachbereiches dienen.

²Standard Query Language - Abfragesprache für relationale Datenbanken

³Parts Library - Implementierung nach ISO-13584

Für diese Abfrage- und Antwortschnittstelle wird der Standard der ISO 29002-31 implementiert, sowie für deren Nutzung gegebenenfalls weitere nötige Standards. Diese Schnittstelle soll einen automatisierten Datenaustausch ermöglichen und technisch auf den Abfrageprozeduren der Datenbankebene (Katalog- und Klassenkonzeptdaten) basieren. Die Implementierung dieser Prozeduren und des Datenbankschemas ist Teil der Arbeit zweier Kommilitonen (Herr Mende und Herr Loth). Die Schnittstelle soll prototypisch implementiert werden und die Machbarkeit und Integration mit der PLIB aufzeigen.

Um das Ziel zu erreichen, soll im Rahmen der Arbeit untersucht werden, welche möglichen sinnvollen Anwendungsfälle sich in der Praxis - basierend auf den zu unterstützenden ISO Standards - ergeben. Dafür ist eine Analyse der Standards nötig, um zu prüfen, ob und in welcher Form die flexiblen Abfragemöglichkeiten unterstützt werden. Wie in Abschnitt 2.1 erwähnt, gibt es auf Datenbankebene oft eine implizite Abfrageschnittstelle, welche in SQL realisiert ist. Es bietet sich daher an, die Möglichkeiten von SQL-Abfragen als Basis für die Untersuchung zu nehmen, und zu prüfen welche Abfragen mit Hilfe des Standards realisiert werden können. Betrachtet werden soll die Projektion und die Selektion.

Projektion Die Attributbeschränkung gleichsam einer »Spaltenauwahl« in relationalen Datenbanken. In SQL das »select« Keyword. In Relationenalgebra:
$$\text{ergebnisprojektion} \leftarrow \Pi_{(\text{attribut}_1, \text{attribut}_2 \dots \text{attribut}_n)}(\text{plib})$$

Selektion Schränkt eine Suche mittels Prädikaten auf passende Tupel ein, gleichsam einer »Zeilenauswahl«. In SQL das »where« Schlüsselwort samt Mengenoperatoren, wie z.B. »UND, ODER, NICHT«. In Relationenalgebra:
$$\text{ergebnisselektion} \leftarrow \sigma_{(\text{praedikat}_1, \text{praedikat}_2 \dots \text{praedikat}_n)}(\text{plib})$$

Für die Schnittstelle ist eine prototypische Implementierung zu erstellen. Weiterer Bestandteil der Arbeit ist unter Beachtung der technischen Vorgaben, eine Beschreibung des Auswahlprozesses der Techniken, Plattform, Architektur und Programmiersprachen. Die Vor- und Nachteile der einzelnen Optionen des Auswahlprozesses und weitere Nutzungs-, respektive Erweiterungs- und Integrationsmöglichkeiten der entwickelten Schnittstelle sind zu erläutern. Die Entwicklung der Software soll nach einem aktuell üblichen Softwareentwicklungsprozess erfolgen.

2.3. Kontext und Abgrenzung

2.3.1. Gesamtkontext der PLIB Abschlussarbeiten

Um einen Überblick zu schaffen, in welchem Kontext sich diese Arbeit befindet, wird nachfolgend eine kurze Übersicht über die PLIB Abschlussarbeiten des Fachbereiches gegeben.

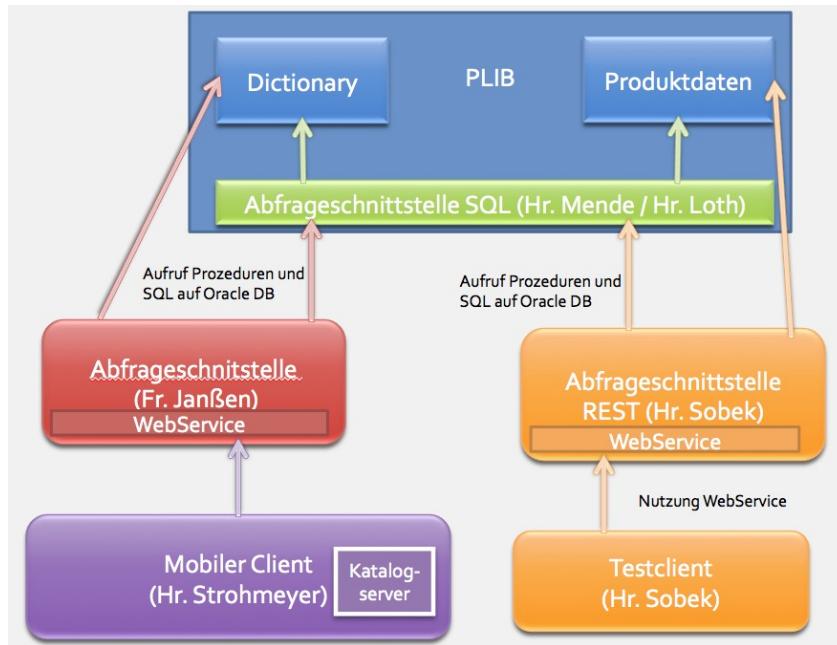


Abbildung 2.1.: Gesamtkontext PLIB Abschlussarbeiten

Die Tabelle 2.1 erläutert die Abbildung 2.1 und listet dazu die jeweiligen Arbeiten und die zur Problemlösung betrachteten ISO-Standards auf.

Bezeichnung	Arbeit von	Erläuterung	ISO Standards
Abfrageschnittstelle als Webservice mit SOAP	Frau Janßen	Implementierung eines Webservices zur Auflösung von Konzept-Identifikatoren in Konzept-Dictionaries/Ontologien	ISO 29002-20
PLIB, Dictionary und Produktdatenbank	Herr Mende und Herr Loth	Meta- und Produktdatenbank sowie eine Abfrageschnittstelle basierend auf Oracle. Entwicklung von Abfrageschnittstellen mit Oracle Prozeduren	ISO 13584-42 (Vergl. ISO909a)
Mobiler Abfrageclient	Herr Lohmeyer	Entwicklung eines mobilen Abfrageclients basierend auf Android. Der Client nutzt die Webservices von Frau Janßen und beinhaltet einen eigenen Katalogserver.	ISO 29002-10, ISO 13584-42
Abfrageschnittstelle als Webservice mit REST für charakteristische Produktdaten	Herr Sobek	Entwicklung einer Schnittstelle zur Abfrage von charakteristischen Produktdaten. Die Schnittstelle wird als RESTful Webservice implementiert.	ISO 29002-31, ISO 29002-10

Tabelle 2.1.: Erläuterung der Gesamtkontextabbildung

2.3.2. Abgrenzung

Die Arbeit umfasst die Implementierung der Use Cases nach Abschnitt 3.3. Dies beinhaltet im wesentlichen den Teil 31 der ISO 29002 - einen Abfragestandard für charakteristische Produktdaten. Weiterhin wird für die Datenübertragung eine Implementierung des Teils 10 der ISO 29002 benötigt, siehe Abbildung 2.2. Die Arbeit beinhaltet nicht die Implementierung eines Identification Guides nach ISO 22745-30. Dieser wird in der Praxis von einem Klienten für eine sinnvolle Vorauswahl der für sich oder seine Organisation benötigten Attribute der Teile verwendet. Jeder Klient definiert für seinen Kontext sinnvolle Attribute und Teiledaten und definiert diese mit Hilfe des Schemas der ISO 22745-30. Dies kann mittels eines Webformulars auf Klientenseite erfolgen oder als allgemeines Formular mit z.B. Excel, welches die für den/die Klienten relevanten Attribute der Produkte, die abgefragt werden sollen, enthält. Für mehr Informationen zum Identification Guide siehe Abschnitt 3.2.

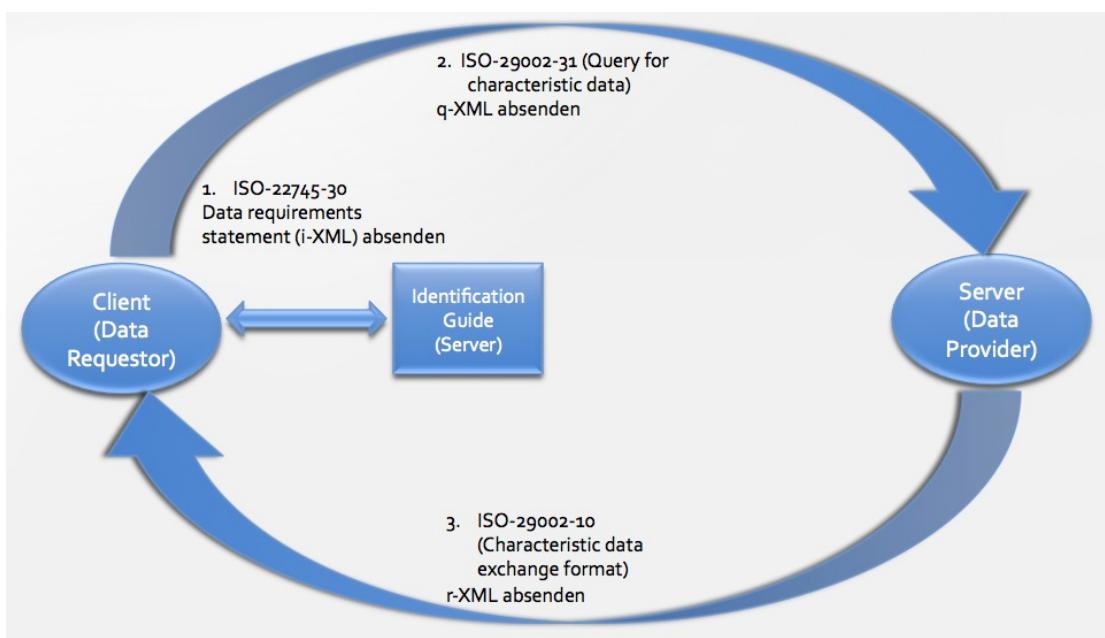


Abbildung 2.2.: Lieferketten⁴

2.3.3. Vorgaben

Für die Implementierung sind folgende nichtfunktionale Anforderungen vorgegeben:

Datenbanksystem Oracle Dieses beinhaltet die PLIB Datenbank samt Abfrageprozeduren und stellt als Teiledatenbank die Basis dar. Dies wird vom Fachbereich bzw. von den Studenten Herr Mende/Herr Loth gestellt.

⁴Abbildung entnommen und abgewandelt aus Benson, Converting Standard Terminology into usable Metadata, 2008 - später in ähnlicher Form auch in Uiterwyk, Die Bedeutung der Merkmalleisten bei eCl@ss, 2012 zu finden.

2. Aufgabenbeschreibung

Webservice Auf Grund der hohen Verbreitung und Integrationsmöglichkeiten soll die Schnittstelle als Webservice entwickelt werden. Die ISO 29002-31 schlägt als Beispiel eine E-Mail-Schnittstelle vor. Siehe Abbildung 2.3, welche besagt:

Transport: not specified in ISO/TS 29002 (could use email) Payload
XML.Query XML schema in ISO/TS 29002-31

Dies ist folglich nur ein Vorschlag.

PLIB Datenbankprozeduren Die vorhandenen Prozeduren zum Zugriff auf die PLIB Datenbank sollen so weit wie möglich verwendet werden.

2.3.4. Datenflüsse

Das System soll einen Webservice zur Verfügung stellen. Dieser Webservice soll eine XML Datei gemäß ISO 29002-31 entgegennehmen. Die entsprechende Verarbeitung des XMLs, sowie die logische Transformation der Anfrage zur Abfrageschnittstelle der Datenbank wird vom System vorgenommen. Die Antwort der Datenbank soll wieder zurücktransformiert werden und als Katalog-XML Datei gemäß ISO 29002-10 zurückgeliefert werden.

Der gerahmte Bereich im unteren Teil der Abbildung 2.3 zeigt in der Datenflussabbildung, was implementiert werden soll. Das Zielsystem, gleichsam das System, welches die Anfrage erhält, wird hier als »Catalogue Server« bezeichnet. Die Kommunikation des Klienten mit dem Location Server, Terminology Server und dem Ontology Server ist Teil der Abschlussarbeit von Fr. Janßen (Vergl. Jan13). Die Abfrageprozeduren des Katalogservers auf Datenbankebene ist Aufgabe von Herrn Mende. Zur Zeit der Abgabe dieser Arbeit, war die Arbeit von Herrn Mende noch in Bearbeitung⁵.

⁵Die Basis dieser Abschlussarbeit ist ein stabiler Implementierungsstand, welcher von Herrn Mende geliefert wurde.

⁶Quelle: entnommen ISO 29002-31 Figure 3 - Major Dataflows. Zur besseren Visualisierung wurde der untere Bereich farblich hervorgehoben.

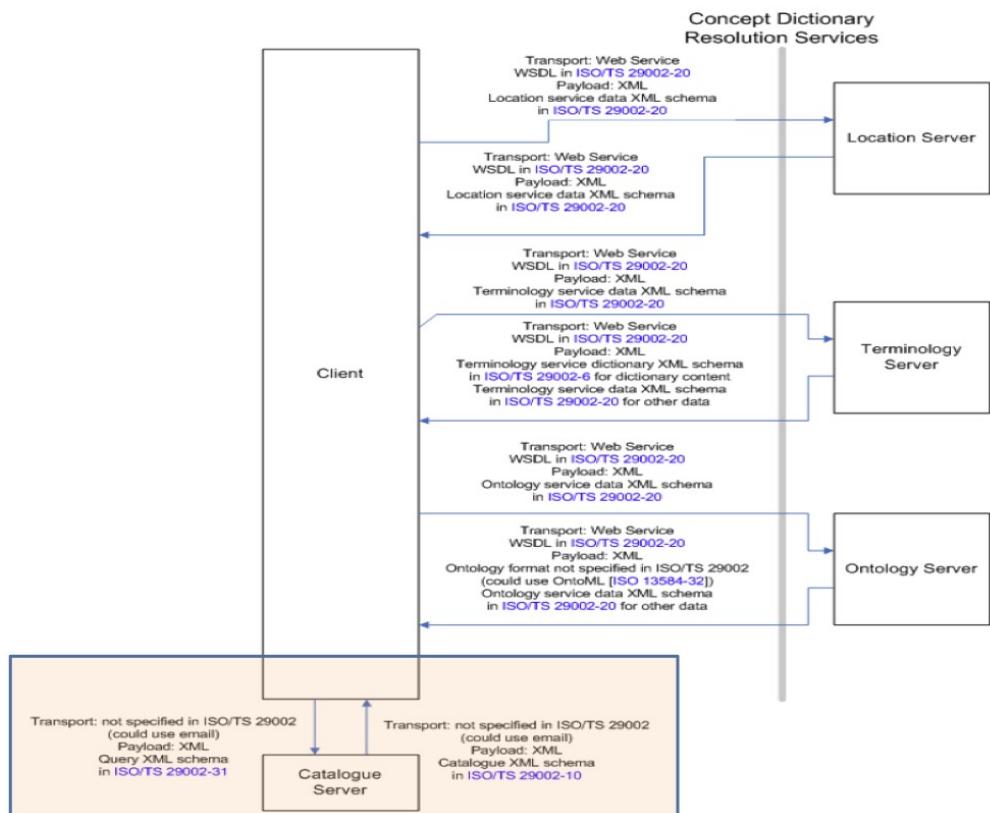


Abbildung 2.3.: Datenflüsse⁶

3. Analyse und Definition der Anforderungen

Vor einer Frage gestellt, die wir vollständig verstanden haben, müssen wir sie von jeder überflüssigen Darstellung abstrahieren, sie auf ihre einfachste Form reduzieren und sie in möglichst kleine Teile zerlegen, die wir dann aufzählen. . .

(René Descartes)

Das Kapitel 2 erwähnt mehrere ISO Standards, welche zur Lösung der Problemstellung unterstützen können. In diesem Kapitel wird der ausgewählte ISO Standard ISO 29002-31 sowie der Standard ISO 22745-30 analysiert, um festzustellen, welche Anforderungen mit Hilfe dieser Standards umgesetzt werden können.

Mittels den Erkenntnissen der Analyse werden Use Cases erarbeitet.

3.1. Analyse ISO 29002-31 - Exchange of characteristic data

Ziel der Analyse ist es, herauszufinden, ob die ISO-Norm zur Erfüllung der Anforderungen ausreichend ist, wie mit der ISO-Norm die Anforderungen umgesetzt werden können und ob gegebenenfalls weitere ISO-Normen zur Erfüllung der Anforderungen betrachtet werden müssen.

Die gesamte Analyse der ISO 29002-31 mit Beschreibung der einzelnen Datencontainer findet sich in Anhang A. Die Analyse zeigt anhand von Beispielen auf, welche Abfragen mit der ISO 29002-31 möglich sind.

Zusammenfassend aus der Analyse ergeben sich folgende Abfragemöglichkeiten:

Simple Query Eine Abfrage nach eindeutigem Identifier eines Konzeptes, mit der Möglichkeit der Einschränkung nach Eigenschaften eines Konzeptes (Projektion). Ferner ermöglicht der Simple Query die Übergabe bekannter Daten eines konkreten Teiles, nach denen gesucht und ein entsprechend passendes Ergebnis geliefert wird. Siehe dazu Unterabschnitt A.1.2.

Parametric Query Ermöglicht die Abfrage nach eindeutigem Identifier eines Konzeptes und zusätzlich die Einschränkung einer Eigenschaft nach konkretem Wert (Selektion). Dies erfolgt mittels Angabe einer characteristic_data_query_expression. Siehe dazu Unterabschnitt A.1.3.

Die konkreten Query Beispiele finden sich dazu in Unterabschnitt A.1.4.

3.2. Analyse ISO 22745-30 - Identification Guide

Ein Identification Guide beschreibt, welche Daten für ein Objekt benötigt werden, damit diese überhaupt sinnvoll für einen bestimmten Zweck eingesetzt werden können. Der Käufer, Produktmanager oder Benutzer definiert die Anforderungen an die Daten. Ein »Datenanforderungsstatement« wird als ein i-xml, eine Identification Guide-xml Datei, erzeugt. Siehe dazu Abbildung 2.2.

Es wird die Frage beantwortet, welche Daten und Eigenschaften zu einem bestimmten Konzept eines Objektes benötigt werden, um den Artikel beispielsweise zu kaufen oder sinnvoll zu verwalten. Diese Anforderungen werden von der Abfrageseite (Kundenseite) definiert, also diejenige, die Daten abfragen möchte (Vergl. Ben11).

Ein Identification Guide referenziert Konzepte eines Dictionaries um Datenanforderungen einer bestimmten Klasse zu beschreiben (Vergl. ISO09b, Kap. 5).

Ein Datenempfänger kann eine Organisation oder eine Gruppe von Organisationen oder Firmen sein, welche ähnliche Datenanforderungen haben. Somit wird eine Identification Guide Gruppe von einer speziellen Organisation verwaltet, welche wiederum selbst Datenempfänger sein kann.

		Bezeichnung Datenfeld					
Conformance class	Nr.	characteristic_data_query_expression	class_ref	property_ref	item	Zu implementieren?	Bemerkung
Parametric Query	1	ja	nein	nein	nein	nein	Ohne class_ref kein Bezug zur Property Referenz möglich. Keine Daten übergeben, nur Selektion, daher Abfrage nicht möglich.
	2	ja	ja	nein	nein	ja	Selektion über class_ref
	3	ja	nein	ja	nein	nein	Ohne class_ref kein Bezug zur Property Referenz möglich.
	4	ja	nein	nein	ja	nein	Ohne class_ref kein Bezug zur Property Referenz möglich. Nur bedingt sinnvoll, wenn die Menge der query_expression eine Teilmenge der item Menge ist. Daher wird dieser Fall ignoriert.
	5	ja	ja	ja	nein	ja	Selektion über class_ref, dann Projektion
	6	ja	ja	nein	ja	nein	Nur bedingt sinnvoll, wenn die Menge der query_expression eine Teilmenge der item Menge ist. Daher wird dieser Fall ignoriert und wie 2 betrachtet.
	7	ja	nein	ja	ja	nein	Ohne class_ref kein Bezug zur Property Referenz möglich. Nur bedingt sinnvoll, wenn die Menge der query_expression eine Teilmenge der item Menge ist. Daher wird dieser Fall ignoriert
	8	ja	ja	ja	ja	ja	Nur bedingt sinnvoll, wenn die Menge der query_expression eine Teilmenge der item Menge ist. Class_ref sollte in items als class_ref vorhanden sein. Falls items vorhanden, wird class_ref ignoriert, daher wird der Fall ignoriert und wie 5 betrachtet
Simple Query	9	nein	nein	nein	nein	nein	Keine Daten übergeben, daher unsinniger Query
	10	nein	ja	nein	nein	ja	Einfacher Query
	11	nein	nein	ja	nein	nein	Keine Daten übergeben nur Property Einschränkung, daher unsinniger Query
	12	nein	nein	nein	ja	ja	Validierung der bekannten Daten bzw. Vervollständigung
	13	nein	ja	ja	nein	ja	Projektion
	14	nein	ja	nein	ja	nein	Bedingt sinnvoll, class_ref sollte in item als class_ref vorhanden sein. Falls items vorhanden, wird class_ref ignoriert, daher wie 12.
	15	nein	nein	ja	ja	ja	Projektion auf Eigenschaften der übergebenen Items
	16	nein	ja	ja	ja	nein	Projektion auf Eigenschaften der übergebenen Items. Bedingt sinnvoll, class_ref sollte in item als class_ref vorhanden sein. Falls items vorhanden, wird class_ref ignoriert, daher wie 15

Abbildung 3.1.: Mögliche Abfragen anhand der Kombination der Daten in XML Query

Es kann somit festgehalten werden, dass für die Erfüllung der Aufgabenbeschreibung nach Kapitel 2 eine Implementierung des Standards ISO 22745-30 nicht notwendig ist, da dieser Standard eine zusätzliche Einschränkung der Konzepte und benötigten Daten auf Anfrageseite beschreibt. Die Abfrage gemäß einer SQL-Selektion und -Projektion ist mit dem Standard 29002-31 möglich. Basierend auf dieser Erkenntnis zeigt Abbildung 3.1 mögliche Abfrageinhalte und bewertet, ob diese umgesetzt werden müssen. Hierbei werden die Attribute »item_description« und »data_specification« nicht betrachtet.

3.3. Anwendungsfälle

Dieser Abschnitt beschreibt mögliche Anwendungsfälle¹ die sich aus ISO 29002-31 ergeben. Es wird beispielhaft ein Use Case aufgelistet und erläutert. Alle weiteren Use Cases sind in Anhang B zu finden.

Die Query-Code-Beispiele sind gekürzt, d.h. es werden die referenzierten Schematamen gemäß XSD nicht aufgeführt.

3.3.1. Was ist ein Use Case?

Alistair Cockburn schreibt dazu (Vergl. Coc00, Kap. 1.1):

»A use case captures a contract between the stakeholders of a system about its behavior. The use case describes the system's behavior under various conditions as it responds to a request from one of the stakeholders, called the primary actor. The primary actor initiates an interaction with the system to accomplish some goal. The system responds, protecting the interests of all the stakeholders. Different sequences of behavior, or scenarios, can unfold, depending on the particular requests made and conditions surrounding the requests. The use case collects together those different scenarios.«

Stakeholder² erwarten folglich ein gewisses Verhalten des Systems unter bestimmten Bedingungen. Dieses Verhalten des Systems in Interaktion mit Akteuren wird durch Use Cases beschrieben.

¹Use Case. Im weiteren Verlauf wird, gerade auch in den Anwendungsfallbeschreibungen von Use Cases gesprochen. Im deutschen Sprachraum ist das englische allgemein bekannte Äquivalent »Use Case« für Anwendungsfall eher geläufig als das deutsche, daher wird darauf verzichtet an manchen Stellen die Begrifflichkeiten einzudeutschen.

²Stakeholder sind Projektbeteiligte, gleichsam Personen oder Institutionen und Dokumente, die in irgend einer Weise vom Betrieb des Systems betroffen sind.

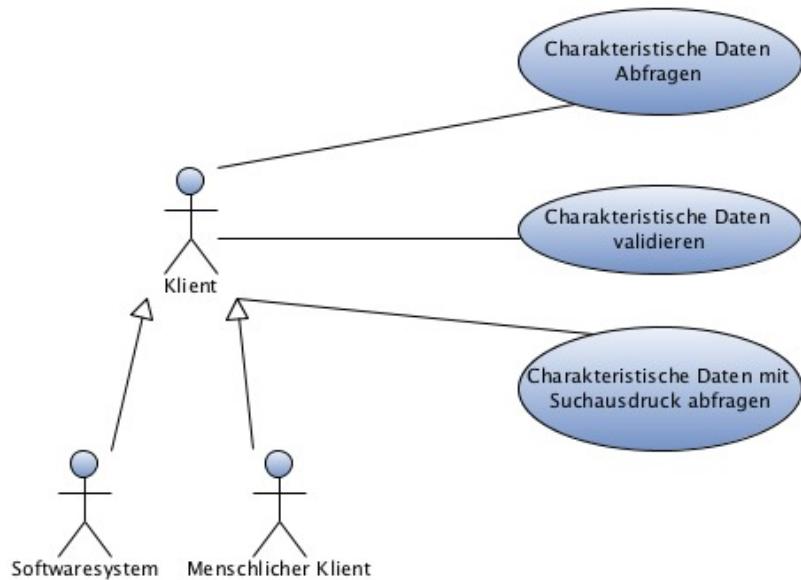


Abbildung 3.2.: Use Case Übersicht

3.3.2. Akteure

Bei der Implementierung geht es um die Erstellung einer Software-Schnittstelle als Webservice. Das bedeutet, dass diese Schnittstelle ohne eine Benutzeroberfläche nicht sinnvoll für einen menschlichen Akteur nutzbar ist. In den nachfolgenden Anwendungsfällen wird vom Akteur »Klient« gesprochen. Der Klient ist allgemein ein Nutzer der Schnittstelle, sei es als menschlicher Akteur welcher über eine Bedienerinterface die Schnittstelle benutzt oder eine direkte Maschinennutzung. Ein Beispiel für eine Maschinennutzung kann eine Anwendung sein, welche automatisiert die Schnittstelle aufruft und Daten abfragt.

3.3.3. Use Case Beschreibungen

Da Use Cases auf verschiedene Weise verfasst werden können, werden hier als Vorlage die Use Case Beschreibungen aus dem Kurs Software Engineering I übernommen, da diese Vorlage sehr übersichtlich, knapp und präzise ist (Vgl. Six09, S. 120ff).

Alle charakteristischen Daten eines Produkts abfragen

use case Charakteristische Daten abfragen

actors

Klient

precondition

Der Klient verwendet einen gültigen Identifier.

main flow

Der Klient gibt einen Identifier (IRDI) eines Konzeptes von Elementen ein und sendet eine Anfrage ab. Die Anfrage wird auf Gültigkeit überprüft. Als Antwort bekommt er ein oder mehrere Datensätze von Teilen mit den entsprechenden charakteristischen Daten³ des Teils mit dem übergebenen Identifier zurück.

postcondition

Alle Daten aller Teile der gewählten Konzepte des Identifiers wurden zurückgegeben.

alternative flow Properties auswählen

Zusammen mit dem Identifier übergibt der Klient einen oder mehrere Property-Identifier und sendet diese erweiterte Anfrage ab.

postcondition

Die mittels Property-Identifier ausgewählten Daten aller Teile der gewählten Konzepte wurden zurückgegeben.

alternative flow Keine Eigenschaftswerte vorhanden

Für das mittels Identifier angefragte Teil sind keine Eigenschaftswerte vorhanden.

postcondition

Es werden keine Daten zurückgeliefert.

exceptional flow Ungültige Identifier oder ungültige Anfrage

Der oder die übergebenen Identifier oder die gesamte Anfrage ist gemäß Spezifikation ungültig.

postcondition

Es wird eine Fehlermeldung zurückgegeben.

end Charakteristische Daten abfragen

Beispiel

Ein Schraubendreher könnte folgendermaßen in einer Produktdatenbank repräsentiert werden:

Identifier 0173-1#01-AAA352#4

Typ Kreuz

Länge 300mm

Spannungsfest ja

Korrekt erweise müsste anstatt der Attribute wie Länge oder Typ ebenfalls ein Identifier stehen. Die Bezeichnungen sind hier zur besseren Lesbarkeit aufgelöst.

Um alle Eigenschaften (Properties) wie »Länge«, »Typ« und »spannungsfest« zu erhalten, muss folgende Abfrage gesendet werden:

³property_values, ISO 29002-10 Kapitel 5.2.4

»Gib mir alle Teile und alle Properties der Klasse mit dem Identifier 0173-1#01-AAA352#4 (Schraubendreher).«

Das Ergebnis ist ein Teil mit allen Attributen (Properties) der gewünschten Konzepte und gegebenenfalls vorhandenen Unterkonzepte. In diesem Falle genau sind es die oben angegebenen Werte.

Die XML-Abfrage sieht wie folgt aus:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <qy:query xsi:schemaLocation="...query query.xsd" xmlns:xsi="http://www.w3.org
   /2001/XMLSchema-instance" xmlns:cat="...catalogue" xmlns:val="...value"
   xmlns:qy="...query" xmlns:bas="...basic">
3   <qy:class_ref>0173-1#01-AAA352#4</qy:class_ref>
4 </qy:query>
```

Listing 3.1: Query Beispiel - alle Daten abfragen

Eine Abfrage, welche als Projektion die Properties der Klasse auswählt, die zurückgeliefert werden sollen, könnte lauten:

»Gib mir alle Teile und die Properties Länge und Typ der Klasse mit dem Identifier 0173-1#01-AAA352#4 (Schraubendreher).«

Das Ergebnis ist ein Teil mit den gewünschten Attributen (Properties).

Die XML-Abfrage lautet:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <qy:query xsi:schemaLocation="...query query.xsd" xmlns:xsi="http://www.w3.org
   /2001/XMLSchema-instance" xmlns:cat="...catalogue" xmlns:val="...value"
   xmlns:qy="...query" xmlns:bas="...basic">
3   <qy:class_ref>0173-1#01-AAA352#4</qy:class_ref>
4
5
6   <!-- identifier von typ und laenge werden uebergeben -->
7   <qy:property_ref>0173-1#01-BBB111#1 0173-1#01-BBB222#1</qy:property_ref>
8
9 </qy:query>
```

Listing 3.2: Query Beispiel - Daten abfragen mit Propertyeinschränkung (Projektion)

Listing 3.2 beinhaltet ein XML-Attribut `property_ref`. Das wird mit einem oder mehreren gewünschten Property Identifiern gefüllt. Werden mehrere Identifier angegeben, so sind diese mit Leerzeichen zu trennen. In diesem Beispiel sind zwei angegeben, das bedeutet, dass diese beiden Eigenschaftswerte abgefragt werden.

3.4. Zusammenfassung des Kapitels

Das Ergebnis der Analyse der Standards ISO 29002-31 und 22745-30 ergibt, dass diese Beschreibungen für die Lösung der Problemstellung, die Entwicklung einer Abfrage- und Antwortschnittstelle analog zur SQL-Selektion und Projektion der Standard ISO

29002-31, ausreichend sind. Sowohl Projektion als auch Selektion ist dadurch möglich. ISO 22745-30 beschreibt als Identification Guide eine weitere klientseitige Einschränkung der abzufragenden Daten. Man stelle sich eine Einschränkung (Definition einer Teilmenge) durch ein Formular vor. Das Formular schränkt die abzufragenden Daten auf die tatsächlich benötigte Menge von Daten gegebenenfalls stark ein und gibt ebenfalls den Typ der Daten vor. Dies wird durch Referenzieren auf Konzepte eines Dictionaries erreicht.

Aus den Anforderungen wurden sinnvolle Use Cases formuliert, welche das Verhalten des Systems beschreiben sollen.

4. System- und Softwareentwurf

A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools. . . .

(*Douglas Adams*)

Dieses Kapitel beschreibt den System- und Softwareentwurf sowie die Auswahl der Umgebung, Plattform, Software, Programmiersprache und der Frameworks.

4.1. Auswahlprozess

Teil der Aufgabe ist es, für das System im Rahmen der nichtfunktionalen Anforderungen eine geeignete Laufzeitumgebung zu schaffen. Dafür sind einige Entscheidungen zu treffen, wie z.B. die Auswahl des Webservices, der Programmiersprache, einzusetzende Frameworks oder der Plattform.

4.1.1. Webservice

Wenn von Webservices gesprochen wird, dann sind meistens SOAP-Webservices gemeint. Allerdings gibt es die sogenannten SOAP-Webservices als auch die RESTful Webservices. Anforderung ist es, einen Webservice zu implementieren. Dazu muss entschieden werden, ob ein SOAP oder RESTful Webservice implementiert werden soll.

Definition

»Web services provide the means to integrate disparate systems and expose reusable business functions over HTTP. They either leverage HTTP as a simple transport over which data is carried (e.g., SOAP/WSDL services) or use it as a complete application protocol that defines the semantics for service behavior (e.g. RESTful services) (S. 2 Dai12)«

SOAP/WSDL Webservice

Frau Janssen hat in ihrer Abschlussarbeit nach ISO 29002-20 einem SOAP/WSDL Webser-vie implementiert (vgl. Jan13). Die ISO 29002-20 verweist in Annex-B auf entsprechende WSDL-Definitionen. An dieser Stelle wird darauf verzichtet, die Einzelheiten eines SOAP/WSDL Webservices zu erläutern und auf Frau Janßens Abschlussarbeit verwiesen (vgl. Jan13, Kap. 3).

Es sei an dieser Stelle kurz erwähnt, dass Webservices, die auf SOAP/WSDL basieren, ein W3C¹ Standard sind. Dies ist häufig auch ein Kriterium, weshalb in der Industrie SOAP/WSDL Webservices stark verbreitet sind.

RESTful Webservice

RESTful Webservices sind per se kein Standard sondern eher ein Programmierparadigma respektive ein Architekturmuster. Stefan Tilkov schreibt dazu in seinem Buch »Rest und HTTP: Einsatz der Architektur des Web für Integrationsszenarien« (S.10 Til09), folgendes:

»[...] Im Rahmen seiner Dissertation - beendet im Jahre 2000 - abstrahierte Fielding² von der konkreten Architektur, die HTTP zugrunde liegt, und legte den Schwerpunkt auf die Konzepte anstatt auf die konkrete Syntax, die Details des Protokolldesigns und die vielen Kompromisse, die aus Kompatibilitätsgründen eingegangen werden mussten. Als Ergebnis entstand der Architekturstil REST. Dieser ist somit eine Stufe abstrakter als die HTTP-Architektur: Theoretisch könnte man die Prinzipien von REST auch mit einem neu erfundenen Satz von Protokollen umsetzen. In der Praxis ist jedoch tatsächlich nur das Web als konkrete Ausprägung des Architekturstils relevant [...]«

(Vgl. auch Fie00)

Fazit

Die Analyse aus Kapitel 3 zeigt, dass eine Abfrage (Query) gemäß Standard als XML dargestellt wird. Diese XML-Dateien müssen folglich an das System gesendet werden. Demnach ist eine Lösung mittels SOAP Webservice als auch RESTful Webservice denkbar. Es wurde ein RESTful Webservice ausgewählt. Die Gründe stellen sich wie folgt dar:

Einfache Implementierung Da RESTful Webservices auf dem HTTP Protokoll basieren und ferner sehr gute und geeignete Frameworks für Java vorhanden sind, (siehe Unterabschnitt 4.1.3) stellt sich für Web Entwickler die Implementierung als einfach heraus. SOAP Webservices sind per se etwas komplexer, da SOAP ein eigenes Protokoll darstellt.

Payload XML Der Payload des Anfrage-Queries wird als XML angegeben (siehe Abbildung 2.3). Als mögliche Übertragung wird E-mail angegeben. Für die Anforderungen dieser Arbeit bedeutet dies eine Implementierung eines Webservices, welcher die XML-Repräsentation des Queries als Payload mittels XML überträgt.

¹World Wide Web Consortium - <http://www.w3c.org>

²Anmerkung des Autors: Roy Thomas Fielding

Folglich könnte sowohl SOAP als auch REST benutzt werden. Voraussetzung ist eine definierte Schnittstelle, welche lediglich ein XML als Payload akzeptiert.

Kein Vorteil bei SOAP Somit hat SOAP keinen Vorteil gegenüber REST, da es in diesem Fall nicht die Operationen selbst anbietet (definiert in der WSDL). Das hat den Nachteil, dass die wohlgeformten vorgegebenen Schemata query.xsd, in der Form nicht genutzt werden können. Es muss somit eine Einbindung in SOAP erfolgen.

Vorteil des Payloads Der klare Vorteil bei der Variante, eine Query XML-Datei gemäß query.xsd Schema des Standards als Payload zu versenden ist der, dass eine Validierungsprüfung der XML gegen vorhandene definierte Regeln des Schemas erfolgen kann (z.B. gültige IRDI), ferner können aus dem Schema passende Modellklassen zur Verarbeitung und Speicherung der Query-Daten in der Applikation generiert werden. Mehr Informationen gibt es in Unterabschnitt 5.3.1.

SOAP Webservice bereits im Einsatz Ein weiterer Grund ist der, dass Frau Janßen in ihrer Abschlussarbeit bereits einen SOAP-Webservice einsetzt. Nennen wir es Vielfältigkeit der Technologien im Fachbereich, jedenfalls bietet es die Möglichkeit eine weitere Technologie zu betrachten, einzusetzen und ggfs. zu vergleichen.

4.1.2. Plattform

Als Laufzeitumgebung kann ein beliebiger Web Container bzw. Web Server ausgewählt werden, welcher Webservices, sei es RESTful Webservices oder SOAP Webservices, unterstützt. Es wurde der Apache Tomcat Server in der Version 7 ausgewählt. Das ist ein üblicher Web Container (Web Server), der mit entsprechenden Frameworks bzw. Bibliotheken sowohl SOAP als auch RESTful Webservices anbieten kann.

Folgende Server sind zum Einsatz ebenfalls möglich:

Jetty <http://www.eclipse.org/jetty/>

Tomcat <https://tomcat.apache.org/>

JBoss <https://www.jboss.org/overview/>

Festzuhalten sei noch, dass es sich beim JBoss Server um einen sogenannten Application Server handelt. Er verfügt zusätzlich zur Web Container Funktionalität noch über weitere Dienste, wie beispielsweise Transaktionsmanagement, Securitymanagement oder verteilte Applikationsunterstützung.

4.1.3. Bibliotheken und Frameworks

Jersey Framework zur Erstellung von RESTful Web Services Jersey

JSF2.0 Komponentenbasiertes Web Framework zur Erstellung von Benutzeroberflächen

Spring Dependency Injection Framework. Bietet darüber hinaus noch weitere Komponenten an. Ausgewählt wurde unter anderem Spring JDBC und Spring Data Oracle, welche einfacheren Zugriff auf relationale Datenbanken sowie auf Prozeduren von relationalen Datenbanken ermöglicht.

4.1.4. Programmiersprache

Vorgegeben ist die Umsetzung eines Webservices. Diese lassen sich in fast allen aktuellen Programmiersprachen entwickeln, folglich kann prinzipiell jede Sprache ausgewählt werden, die Webservices anbieten kann.

Es wurde die Sprache Java gewählt, da diese zum einen in der aktuellen Industrie stark verbreitet und zum anderen der Autor dieser Arbeit seit vielen Jahren damit vertraut ist. Ferner besteht hier eine Abhängigkeit zur Auswahl der Plattform gleichsam des Web Containers, siehe Unterabschnitt 4.1.2. Beim Einsatz von Java ist man an einen Web Container (Web Server) gebunden, welcher auch Java unterstützt.

Ein weiterer Aspekt ist, dass Software, welche mit Java entwickelt wurde, auf vielen Betriebssystemen lauffähig und somit portierbar ist. Dies ist zwar keine Anforderung des Projektes, ermöglicht allerdings die Arbeit und Entwicklung in beliebigen Systemen.

4.2. Softwaredesign und Architektur

4.2.1. Bausteinsicht

Die Bausteinsicht bildet die Aufgaben des Systems auf Software-Bausteine oder -Komponenten ab (S. 98ff Sta09).

Es soll mit Hilfe dieser Sichten ein Überblick über den Aufbau des Systems und den Abhängigkeiten der einzelnen Komponenten geschaffen werden. Dazu wird das System im top-down Ansatz aufgezeigt und verfeinert.

4.2.2. Kontextabgrenzung - Systemüberblick und angrenzende Systeme

Die Kontextabgrenzung in Abbildung 4.1 zeigt das System im Zusammenspiel mit anderen Systemen und zeigt die Systemgrenzen auf.

Klient Der Klient stellt den Nutzer des Query Services dar. Er erzeugt das XML File, welches als Query an den Service geschickt wird. Der Transport erfolgt über das HTTP Protokoll.

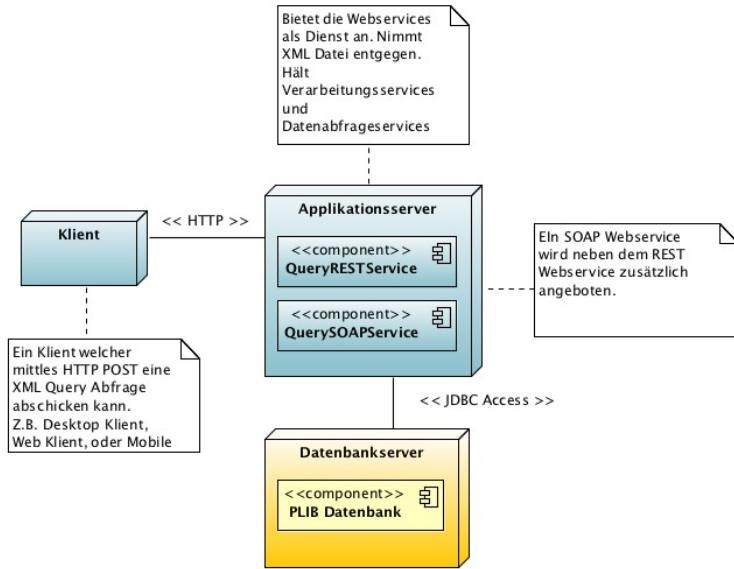


Abbildung 4.1.: Bausteinsicht - Kontextabgrenzung

Applikationsserver Der Applikationsserver ist der Hauptbaustein. Dieser Baustein enthält alle entwickelten Komponenten. Sichtbar von außen ist der QueryRESTService, dieser Service ist ein REST Webservice und nimmt XML-Dateien als Payload eines POST Requests entgegen. Zusätzlich zum REST Webservice wird ein SOAP Webservice angeboten namens QuerySOAPService.

Datenbankserver Der Datenbankserver beinhaltet die PLIB Datenbank mit den entsprechenden Zugriffsprozeduren auf die Daten.

4.2.3. Level 1 - Plib characteristic query

Die Bausteinsicht Level 1 in Abbildung 4.2 zeigt alle Komponenten des entwickelten Systems auf und deutet die externen Schnittstellen an. Mittels «use» Beziehungen erkennt man die Abhängigkeiten der einzelnen Komponenten.

QueryRESTService Der Zweck dieser Komponente ist das Entgegennehmen des HTTP-Requests (Query-XML File), das Weiterleiten an die weiterverarbeitenden Komponenten und letztlich das Zurücksenden der Rückantwort (Katalog-XML). Bietet nach außen die Webservice-Schnittstelle an.

QuerySOAPService Der Zweck dieser Komponente ist das Entgegennehmen des SOAP-Requests (SOAP File gemäß SOAP Spezifikation mit eingebundenen Schema-ta), das Weiterleiten an die weiterverarbeitenden Komponenten und letztlich das Zurücksenden der Rückantwort (Katalog-XML). Bietet nach außen die Webser-viice-Schnittstelle an. Der SOAP-Service muss nicht explizit die Unmarshalling-Komponente aufrufen, um das XML-File in ein entsprechendes Modell zu über-führen. Dies geschieht implizit.

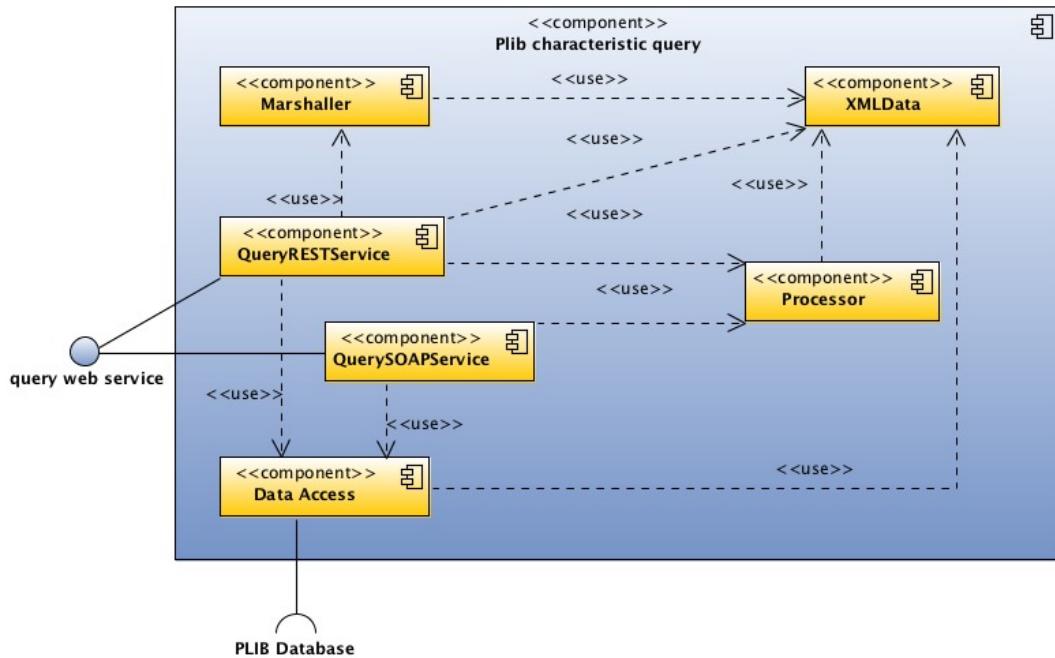


Abbildung 4.2.: Bausteinsicht - Level 1

Data Access Diese technische Komponente beinhaltet die Zugriffsschicht auf die externe Datenbankschnittstelle und bietet entsprechend vereinfachte Abfrageschnittstellen für die anderen Komponenten an.

Marshaller Eine weitere technische Komponente. Diese ist für das Einlesen und Validieren der eingegangenen Query-XML Datei verantwortlich. Ferner transformiert diese Komponente die Informationen aus der Query-XML nach Validierung in das im System benutze Datenmodell aus der Komponente XMLData.

XMLData Beinhaltet das Datenmodell des Systems. Sowohl die eingehenden Query-XML Daten, als auch die ausgehenden Katalog-XML Daten werden intern in ein entsprechendes Modell zur Verarbeitung abgelegt, sodass darauf gearbeitet werden kann.

Analyser Diese Komponente ist für die Analyse des Queries, welches bereits als Modell vorliegt, verantwortlich. Erkennt, ob es sich um einen »Simple Query« oder um einen »Parametric Query« handelt und leitet den Query zur Weiterverarbeitung an die Handler-Komponente weiter.

Handler Erhält den Query vom Analyser und verarbeitet gemäß der Erkennung vom Analyser den Query weiter. Dazu werden die Querydaten gemäß des Datenbankmodells transformiert und die Data Access Komponente aufgerufen. Die Antwort der Data Access Komponente wird in das entsprechende Modell für die Katalogdaten umgewandelt und zurück an den Handler geleitet.

Weitere verfeinerte Sichten aller Bausteine aus Abbildung 4.2 sind in Anhang H aufgeführt.

5. Implementierung

Es ist nicht genug, zu wissen, man muß auch anwenden; es ist nicht genug, zu wollen, man muß auch tun. . .

(Johann Wolfgang von Goethe)

5.1. Configuration Management und Setup

5.1.1. Versionsverwaltung

»A version control system (VCS) is a methodology or tool that helps you keep track of changes you make to the files in your project. In its simplest, manual form, a VCS is you creating a copy of the file you're working with and adding the date and time to the end of it. (S. 15 Swi08)«

Ein Versionskontrollsystem sollte immer in der Softwareentwicklung eingesetzt werden, um einerseits Änderungen nachvollziehen zu können, allerdings auch und vor allem, um Änderungen rückgängig machen zu können, die ggf. fälschlicherweise durchgeführt wurden. Ferner ist es möglich, verschiedene Versionen der Software gleichsam des Quellcodes zu führen und zu verwalten. Außerdem ermöglichen aktuelle Versionskontrollsysteme die Zusammenarbeit mehrerer Entwickler am Quellcode. Hierbei hilft die Wiederherstellbarkeit und Nachvollziehbarkeit der Änderungen.

5.1.2. Versionsverwaltung mit GIT

Was ist GIT?

GIT ist ein Versionskontrollsystem, kurz VCS, GIT ist darüberhinaus ein verteiltes Versionskontrollsystem, kurz DVCS.

»Distributed version control systems (DVCSs) are no different in that respect. Their main goal is still to help us track changes we make to the projects we're working on. The difference between VCSs and DVCSs is how developers communicate their changes to each other. (S. 15 Swi08)«

In diesem Projekt ist kein verteiltes Versionskontrollsystem nötig, da es nur einen Bearbeiter der Daten und Quelltexte gibt. Es wurde dennoch gewählt, da es zum einen genau wie ein »normales«, folglich nicht verteiltes Versionskontrollsystem eingesetzt werden kann, der Autor mit diesem System sehr vertraut ist und ggf. zukünftig auf diese Arbeit aufbauende oder weiterführende Projekte mit diesem System einfach weiterarbeiten können.

5. Implementierung

Die \LaTeX Quelltexte dieser Arbeit sind bei einem frei verfügbaren Versionskontrollsystemanbieter untergebracht. Das Repository kann mit folgendem Befehl geklont werden (»username« durch Benutzernamen beim Anbieter Bitbucket ersetzen):

```
git clone https://username@bitbucket.org/ssobek/masterarbeit.git
```

Der Quellcode der Implementierung kann mit folgendem Befehl geklont werden:

```
git clone https://username@bitbucket.org/ssobek/plib-query.git
```

Klonen bedeutet bei verteilten Versionskontrollsystemen, das gesamte Repository zu kopieren. Es kann dann auf diesem Repository lokal gearbeitet werden.

Für mehr Informationen über GIT und den Funktionsumfang sei auf entsprechende Literatur verwiesen (vgl. Swi08).

5.1.3. Build Management mit Maven

Was ist Maven?

»Maven ist ein deklaratives Build Management System. Das heißt, es wird lediglich der Inhalt des Projekts beschrieben, nicht die Struktur oder die Abläufe, die zur Kompilierung und Veröffentlichung notwendig sind. Die Philosophie hinter Maven heißt ›Konvention über Konfiguration‹ - Strukturen müssen nicht definiert werden, sondern sind vorgegeben. So wie die Projekt- und Verzeichnisstruktur ist auch die Reihenfolge der Arbeitsschritte vorgegeben, die Maven ausführt, um ein Projekt zu bauen. (S. 27 Spi11)«

Wozu ist Maven in diesem Projekt nötig?

Maven ermöglicht konkret in diesem Projekt dem Betreuer, Professor oder Studenten, deren Arbeiten ggf. aufbauend auf diese Arbeit sind, das Projekt standardisiert, einfach und ohne Kenntnisse von Maven oder anderen Werkzeugen zu erstellen. Maven findet dazu alle benötigten Abhängigkeiten automatisiert gemäß Konfiguration und erzeugt das Projekt. Es werden gemäß des standardisierten Buildablaufes von Maven mehrere Phasen durchlaufen, die beispielsweise sicherstellen, dass erst alle automatischen Tests erfolgreich durchlaufen sein müssen, bevor das Endprodukt erzeugt wird.

In diesem Projekt wird Apache Maven in der Version 3 eingesetzt.

Für mehr Informationen sei auf entsprechende Literatur verwiesen (vgl. Spi11).

5.1.4. Apache Tomcat

Als Web Container wird der Apache Tomcat in der Version 7 eingesetzt. Die Installation wird in Anhang C beschrieben.

5.1.5. Tests

Einer der wichtigsten Punkte in der Softwareentwicklung ist das Testen. Es werden ausschließlich Entwicklertests betrachtet. Hier gibt es zwei Bereiche, automatische Entwicklertests und manuelle Entwicklertests.

Automatische Entwicklertests mit JUnit

Für die automatischen Entwicklertests wird ein Framework namens JUnit eingesetzt. Der Vorteil ist, dass dieses Framework weit verbreitet, bekannt und ausgereift ist. Es gibt zwei Arten von Tests. Die Unit Tests und die Integrationstests.

Unit Tests testen eine Komponente isoliert von allen abhängigen Komponenten. Das bedeutet, dass nur diese Komponente auf die korrekte Funktionsweise hin getestet wird. Jegliche Abhängigkeiten werden durch Mock- oder Fakeobjekte ersetzt, sodass ein genaue Testumgebung für diese Komponente simuliert werden kann.

Integrationstests testen mehrere Komponenten im Verbund und Anwendungsszenarien. Ein Beispiel ist ein Test, welcher auf der grafischen Benutzeroberfläche einsetzt, dort einen Knopf drückt (z.B. Benutzer anlegen) und durch alle Schichten hindurch bis hin zur Datenbank einen Eintrag in der Datenbank anlegt. Anschließend wird entweder in der Datenbank oder erneut auf der Benutzeroberfläche geprüft (Liste aller Benutzer), ob der Eintrag erzeugt wurde. Dies ist ein Beispiel, wo die Gesamtapplikation integriert und somit lauffähig sein muss, damit man dieses Szenario testen kann. Ein Integrations-test ist es bereits schon dann, wenn automatisiert ein Datenzugriffsobjekt im Verbund mit der Datenbank getestet wird, da hier bereits mehrere Komponenten zusammen getestet werden.

Mehr Informationen und einige Code-Beispiele in Anhang D.

Manuelle Entwicklertests

Bei manuellen Entwicklertests werden Testcases aus den Anforderungen abgeleitet. Diese Tests sind sogenannte Black Box Tests. Das bedeutet, dass das Objekt, welches getestet wird als Black Box betrachtet wird. Es sind keine technischen Details über Aufbau dieses Testobjektes bekannt und es werden keine Annahmen getroffen. Lediglich die Schnittstelle nach Außen wird betrachtet und gemäß Spezifikation der Ein- und Ausgangsdaten getestet.

Nachfolgend ist ein manueller Testfall aufgeführt, welcher eine ungültige IRDI (gemäß XSD ungültig) übermittelt und einen Fehler erwartet.

Vorbedingungen • Die Datenbankverbindung aufbauen: Oracle Datenbank muss gestartet werden.

Eingabedaten Testdatei simple_query_illegal_irdi.xml.

Durchführung XML-Datei wird mittels curl Befehl an den Server gesendet: curl -v -H 'Content-Type: application/xml' -X POST -data '@simple_query_illegal_irdi.xml' http://localhost:8080/plib-characteristic-query/rest/ws/query

Erwartetes Ergebnis Fehlermeldung

Tatsächliches Ergebnis Fehlermeldung, marshalling error

OK/Nicht OK? OK

Alle weiteren Testfälle und Ergebnisse sowie die Beschreibung einiger Testtools zum Ausführen der manuellen Tests finden sich in Anhang E.

5.2. Webservice

Wie in Unterabschnitt 4.1.1 erwähnt, wurde entschieden einen RESTful Webservice zu erstellen. Dafür wurde das Jersey-Framework ausgewählt.

Zur Erklärung Java unterstützt REST durch die Java Specification Request 311 (JSR)¹.

Diese Spezifikation wird JAX-RS - die Java API for RESTful Webservices genannt.

Jersey ist die Referenzimplementierung, bereits in Java 6 integriert. Das ist auch der Grund, weshalb die Jersey Implementierung als Basis gewählt wurde.

Folgende Schritte sind notwendig um einen RESTful Webservice mit Jersey zu erstellen:

5.2.1. Servlet Konfiguration in web.xml

In die Konfigurationsdatei web.xml des Webcontainers (Apache Tomcat) muss das Servlet für Jersey hinzugefügt werden, sodass Webservice-Anfragen an dieses Servlet möglich sind.

Listing 5.1 zeigt den Ausschnitt aus der web.xml des PLIB-Projektes.

```
1  <servlet>
2      <servlet-name>jersey-servlet</servlet-name>
3      <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</
4          servlet-class>
5      <init-param>
6          <param-name>com.sun.jersey.config.property.packages</param-name>
7          <param-value>de.feuplib.webservice.rest</param-value>
8      </init-param>
9      <load-on-startup>1</load-on-startup>
</servlet>
```

Listing 5.1: Jersey Servlet Konfiguration in web.xml

¹Spezifikation: <http://jcp.org/aboutJava/communityprocess/mrel/jsr311/index.html>

Ferner muss in der web.xml ein sogenannter »Mappingeintrag« angelegt werden. Hierdurch wird dem Webserver mitgeteilt, welche URL-Aufrufe an welches Servlet zur Verarbeitung geleitet werden sollen.

```

1 <servlet-mapping>
2   <servlet-name>jersey-servlet</servlet-name>
3   <url-pattern>/rest/*</url-pattern>
4 </servlet-mapping>
```

Listing 5.2: Jersey Servlet Mappingkonfiguration in web.xml

Das Konfiguration Beispiel in Listing 5.2 besagt, dass das Servlet mit dem Namen »jersey-servlet«, welches im Beispiel Listing 5.1 konfiguriert wurde, alle Anfragen mit der URL »/rest/*« entgegennehmen soll. Das Muster »/rest/*« bedeutet, dass beliebige URLs nach »/rest/« akzeptiert werden. Zum Beispiel: »/rest/webservice« oder »/rest/service/name«.

Unter der Annahme, dass die Applikation auf dem lokalen Rechner installiert wurde und auf Port 8080 lauscht, der Applikationskontext² »plib-characteristic-query« ist, ergibt sich als aktuelle Gesamt-URL für den Webservice der Applikation »http://localhost:8080/plib-characteristic-query/rest/«.

5.2.2. Webservice Klasse

Der Einstiegspunkt für den Webservice ist eine Klasse. Eine Applikation kann mehrere solcher Einstiegspunkte haben. Damit die Navigation von der URL der Anfrage zur entsprechenden Klasse funktioniert, wird jede Klasse mittels Annotation markiert und ein weiterer Pfad-Präfix definiert. Das Beispiel Listing 5.3 zeigt, dass mittels @Path der Suffix /ws definiert wird. Das bedeutet, dass die Klasse QueryService den URI-Pfad /ws zugewiesen bekommt.

```

1 ...
2 @Path("/ws")
3 public class QueryService {
4 ...
```

Listing 5.3: Jersey Webservice Klasse

Somit ergibt sich als aktuelle Gesamt-URL für den Webservice der Applikation »http://localhost:8080/plib-characteristic-query/rest/ws«.

Der nächste Schritt ist, die entsprechende Methode zu definieren, welche die Anfrage final entgegennimmt und verarbeitet (siehe Listing 5.4).

```

1 @POST
2 @Path("/query")
3 @Consumes("application/xml")
4 @Produces(MediaType.APPLICATION_XML)
5 public String query(String queryXML) {
6     LOGGER.info("Incoming query XML content :" + queryXML);
```

²Die Ausführungsumgebung einer Software Applikation innhalb eines Applikationsservers

```
7     QueryType queryType = unmarshal(queryXML);
8     LOGGER.info("QueryType: " + queryType);
9     CatalogueType catalogue = queryPipe.filter(queryType);
10
11    LOGGER.info("Filled Catalogue: " + catalogue);
12    String marshalledCatalogue = marshall(catalogue);
13
14    LOGGER.info("Marshalled catalogue: " + marshalledCatalogue);
15    return marshalledCatalogue;
16 }
```

Listing 5.4: Jersey Methode

Die Konfiguration der Methode wird über Annotation vorgenommen. Nachfolgend die Erklärung der Annotations aus Listing 5.4:

@POST Definiert die HTTP-Methode. Hier POST. Einige weitere Möglichkeiten des HTTP-Protokolls sind GET, PUT und DELETE.

@Path('/query') Definiert den URL-Pfad Suffix für diese Methode. Um sie als Webservice via HTTP aufzurufen, lautet die finale URL:
»<http://localhost:8080/plib-characteristic-query/rest/query>«.

@Consumes('application/xml') Definiert den MIME-Type³, welcher von diesem Service (dieser Methode) konsumiert werden kann. Wird ein anderer Typ als POST an diesen Service geliefert, weist der Service die Anfrage ab.

@Produces(MediaType.APPLICATION_XML) Definiert den MIME-Type des Inhaltes, der vom Service als Antwort zurückgeliefert wird.

5.3. Query-Verarbeitung

Der Webservice, welcher in Abschnitt 5.2 beschrieben ist, nimmt in der Applikation das Query-XML-File entgegen. Die Struktur des XML-Files ist durch die query.xsd vorgegeben (ISO09c, 27).

Der nächste Schritt ist es, diese XML-Datei zu verarbeiten. Dazu muss sie geparsed und die Informationen des Queries in ein entsprechendes Modell überführt werden. Diesen Prozess nennt man Unmarshalling.

Die folgenden Schritte müssen für das Unmarshalling durchgeführt werden:

1. Ein Modell in Java erstellen
2. XML parsen und in das Modell überführen
3. Validierung des Modells gemäß der Regeln aus der Schema-Datei

³Internet Media Type oder auch Content-Type.

5.3.1. Modell-Generierung

Ein entsprechend valides Modell anhand der XSD manuell in Java aufzubauen, wäre sehr mühsam und fehleranfällig. Java liefert mit der JAXB Bibliothek die Möglichkeit, die Modellklassen von Java aus den XSDs zu generieren.

Benötigte XSD-Dateien

Die XSD-Dateien der ISO 29002-31 wurden freundlicherweise von Dr. Gerry Radack von der ECCMA zur Verfügung gestellt.

Die query.xsd referenziert die weiteren folgenden Schema-Dateien:

- basic.xsd
- catalogue.xsd
- identifier.xsd
- value.xsd

Generierung mit JAXB

Die Generierung startet man als Java-Kommando in der Konsole, dazu ist folgender Befehl abzusetzen:

```
xjc query.xsd -d plib-characteristic-data/
```

Problem Namensräume

Bei der Generierung wurde die folgende Fehlermeldung angezeigt:

```
»[ERROR] The package name iso.std.iso.ts._29002._-31.ed_1.tech.xml_schema.query
used for this schema is not a valid package name. line 18 of file query.xsd«
```

Der Grund für den Fehler ist, dass in der query.xsd die Namensräume⁴ wie in Listing 5.5 definiert sind. Da JAXB die Namespaces nutzt, um die entsprechenden Paketstrukturen für die Modelle in Java zu erstellen, schafft es JAXB nicht, diese in die entsprechende valide Form umzuwandeln. Man erkennt es daran, dass aus

```
urn:iso:std:iso:ts:29002:-31:ed-1:tech:xml-schema:query
```

in der Fehlermeldung

```
iso.std.iso.ts._29002._-31.ed_1.tech.xml_schema.query
```

wird⁵. Der Bindestrich vor der 31 ist nicht als erstes Zeichen eines Unterpakettamens erlaubt. Wenngleich man hier erwarten würde, dass dies abgefangen und entsprechend der Umwandlungsregeln in valide Bezeichner konvertiert wird, so führt eine Benamung

⁴Im weiteren Verlauf wird der englische Begriff Namespaces für Namensräume benutzt.

⁵Java Paketnamen enthalten als Trenner zwischen den Ebenen einen Punkt. Valide wäre z.B. iso.std.iso.ts.query

5. Implementierung

der Pakete nach den Namespaces zu unübersichtlichen und nicht aussagekräftigen Paketstrukturen.

```
1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2   xmlns:qy="urn:iso:std:iso:ts:29002:-31:ed-1:tech:xml-schema:query"
3   xmlns:cat="urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:catalogue"
4   xmlns:val="urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:value"
5   xmlns:bas="urn:iso:std:iso:ts:29002:-4:ed-1:tech:xml-schema:basic"
6   xmlns:id="urn:iso:std:iso:ts:29002:-5:ed-1:tech:xml-schema:identifier"
7   targetNamespace="urn:iso:std:iso:ts:29002:-31:ed-1:tech:xml-
8     schema:query" elementFormDefault="qualified">
9     <xs:import namespace="urn:iso:std:iso:ts:29002:-4:ed-1:tech:xml-schema:basic"
10    schemaLocation="basic.xsd"/>
11     <xs:import namespace="urn:iso:std:iso:ts:29002:-5:ed-1:tech:xml-
12       schema:identifier" schemaLocation="identifier.xsd"/>
13     <xs:import namespace="urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-
14       schema:catalogue" schemaLocation="catalogue.xsd"/>
15     <xs:import namespace="urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:value"
16       schemaLocation="value.xsd"/>
17     ...
18   </xs:schema>
```

Listing 5.5: query.xsd Namespace Definitionen

Für dieses Problem gibt es zwei Lösungsoptionen. Entweder die Namespace-Definitionen aller Namespaces in den XSD-Dateien anpassen oder die XSD-Dateien in der Form belassen und einen programmatischen Weg finden, um die Namespaces umzudefinieren.

Das Anpassen aller XSD-Dateien hat zwei große Nachteile:

1. Es ist aufwändig und fehleranfällig alle Dateien anzupassen. Diese haben - wie in Listing 5.5 zu sehen - Abhängigkeiten untereinander⁶.
2. Änderungen an der lokalen Schemadatei machen mögliche spätere Integrationen einer neuen XSD-Version des ISO-Komitees schwierig.

Zu bevorzugen ist eine konfigurierbare Generierung. JAXB ermöglicht mit einem sogenannten Binding-File⁷, die Namespaces abzuändern. Listing 5.6 zeigt die Konfiguration. Die Generierung kann mit dem Shell-Befehl

```
xjc -b binding.xjb -d gen-src query.xsd
```

gestartet werden.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <jaxb:bindings xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
3   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4   xmlns:xjc="http://java.sun.com/xml/ns/jaxb/xjc"
5   jaxb:version="2.0">
6   <jaxb:bindings schemaLocation="query.xsd" node="/xsd:schema">
7     <jaxb:schemaBindings>
8       <jaxb:package name="de.feuplib.xml.query" />
```

⁶Die Abhängigkeiten erfolgen über die xs:import Anweisungen. Damit werden weitere Schema-Dateien eingebunden und die entsprechenden Elemente nutzbar gemacht.

⁷Ein Binding-File ist eine XML-Datei, welche gemäß JAXB eine Konfiguration für die Generierung ermöglicht.

```

9      </jaxb:schemaBindings>
10     </jaxb:bindings>
11     <jaxb:bindings schemaLocation="basic.xsd" node="/xsd:schema">
12       <jaxb:schemaBindings>
13         <jaxb:package name="de.feuplib.xml.basic" />
14       </jaxb:schemaBindings>
15     </jaxb:bindings>
16     <jaxb:bindings schemaLocation="catalogue.xsd" node="/xsd:schema">
17       <jaxb:schemaBindings>
18         <jaxb:package name="de.feuplib.xml.catalogue" />
19       </jaxb:schemaBindings>
20     </jaxb:bindings>
21     <jaxb:bindings schemaLocation="identifier.xsd" node="/xsd:schema">
22       <jaxb:schemaBindings>
23         <jaxb:package name="de.feuplib.xml.identifier" />
24       </jaxb:schemaBindings>
25     </jaxb:bindings>
26     <jaxb:bindings schemaLocation="value.xsd" node="/xsd:schema">
27       <jaxb:schemaBindings>
28         <jaxb:package name="de.feuplib.xml.value" />
29       </jaxb:schemaBindings>
30     </jaxb:bindings>
31
32     <jaxb:globalBindings>
33       <!-- let the classes implement serialisable -->
34       <jaxb:serializable uid="1" />
35       <!-- let the classes extend own abstract class for providing some extra
            functionality for each one -->
36     </jaxb:globalBindings>
37   </jaxb:bindings>

```

Listing 5.6: Binding File binding.xjc

Wie man erkennen kann, wird für jede Schema-Datei ein eigener Paketname definiert. Das hat den Vorteil, dass die einzelnen Datentypen aus den jeweiligen XSD-Dateien passend in eigene Pakete generiert werden und nicht alle in ein Verzeichnis. Das ist deutlich übersichtlicher.

Die Modell-Dateien werden folgendermaßen abgelegt:

query.xsd de.feuplib.xml.query

identifier.xsd de.feuplib.xml.identifier

basic.xsd de.feuplib.xml.basic

value.xsd de.feuplib.xml.value

catalogue.xsd de.feuplib.xml.catalogue

Zeile 6-10 aus Listing 5.6 zeigt auf, wie im Binding-File ein anderer Paketname für die Datei query.xsd definiert werden kann.

5.3.2. Einbinden in Buildprozess mit Maven

Da als Build-Werkzeug Maven verwendet wird, kann der gesamte Generierungsprozess darüber abgebildet werden.

Sepаратes Source-Verzeichnis

Die Standardprojektform eines Maven-Projektes hat einen sogenannten Source-Verzeichnis (src). Darin befindet sich ein main-Verzeichnis. Dort werden die Klassen der Applikation abgelegt. Ferner beinhaltet das src-Verzeichnis ein test-Verzeichnis. Darin werden Testklassen abgelegt. Damit die generierten Sourcen klar von den anderen Source-Dateien getrennt sind, soll ein separates Source-Verzeichnis angelegt werden. Der weitere Vorteil ist, dass dieses Verzeichnis beispielsweise auch vom Versionskontrollsystem ausgenommen werden kann, da diese Klassen während des Build-Prozesses jeweils generiert werden und somit keiner Versionierung bedürfen. Das wäre aufwändiger zu realisieren, wenn die Dateien exakt im gleichen Source-Folder generiert würden wo alle anderen Source-Dateien liegen⁸.

Maven bietet mittels des Plugins »build-helper-maven-plugin« die Möglichkeit, dies zu erzeugen. Die generierten Sourcen der XSD-Dateien werden in ein separates Verzeichnis namens »generated« erzeugt (siehe Listing 5.7).

```
1 <plugin>
2   <groupId>org.codehaus.mojo</groupId>
3   <artifactId>build-helper-maven-plugin</artifactId>
4   <executions>
5     <execution>
6       <id>add-source</id>
7       <phase>generate-sources</phase>
8       <goals>
9         <goal>add-source</goal>
10      </goals>
11      <configuration>
12        <sources>
13          <source>src/main/generated</source>
14        </sources>
15      </configuration>
16    </execution>
17  </executions>
18</plugin>
```

Listing 5.7: Build Helper Maven Plugin

JAXB Maven Plugin

Um JAXB mit Maven zu nutzen, muss das Maven-Plugin »maven-jaxb2-plugin« in die pom.xml eingetragen werden. Listing 5.8 zeigt den XML-Ausschnitt aus der pom.xml.

```
1 <plugin>
2   <groupId>org.jvnet.jaxb2.maven2</groupId>
3   <artifactId>maven-jaxb2-plugin</artifactId>
4   <version>0.8.0</version>
5   <configuration>
```

⁸Hinweis: Moderne Versionskontrollsystme wie das eingesetzte GIT, ermöglicht es, nach Mustern Dateien oder Ordner von der Versionierung auszunehmen. Allerdings müsste das auf Paketbasis erfolgen, sozusagen ab de.feuplib.xml.*., das macht die Umgebung im Ganzen komplexer.

```

6      <schemaDirectory>src/main/resources/schema</schemaDirectory>
7      <generateDirectory>src/main/generated</generateDirectory>
8      <removeOldOutput>true</removeOldOutput>
9  <!-- we do not use bindingDirectory as if we put the binding.xjb in the schema
   directory it will be taken -->
10 <!--
       <bindingDirectory>src/main/resources/binding</
           bindingDirectory> -->
11
12 <!-- Setting the generated package in pom will override what you set in binding.
   xjb file, thus commented out -->
13 <!--
       <generatePackage>de.feuplib.jaxb</generatePackage> -->
14       <strict>false</strict>
15       <extension>true</extension>
16       <plugins>
17         <plugin>
18           <groupId>org.jvnet.jaxb2_commons</groupId>
19           <artifactId>jaxb2-basics</artifactId>
20           <version>0.6.2</version>
21         </plugin>
22         <plugin>
23           <groupId>org.jvnet.jaxb2_commons</groupId>
24           <artifactId>jaxb2-basics-annotate</artifactId>
25           <version>0.6.2</version>
26         </plugin>
27       </plugins>
28       <args>
29         <arg>-Xannotate</arg>
30         <arg>-XtoString</arg>
31       </args>
32     </configuration>
33     <executions>
34       <execution>
35         <id>generate</id>
36         <goals>
37           <goal>generate</goal>
38         </goals>
39       </execution>
40     </executions>
41   </plugin>

```

Listing 5.8: JAXB Maven Plugin

5.4. Transformation und Abfrage der PLIB-Prozeduren

Die eingehende Query-Nachricht wird wie in Abschnitt 5.3 beschrieben in ein Java Modell umgewandelt. Anschließend müssen diese Daten für die Abfrage zur Datenbank gemäß der PLIB-Prozeduren transformiert werden. Die Anforderung ist es, so weit wie möglich die vorhandenen Prozeduren der PLIB zu nutzen.

Die Prozeduren der PLIB nehmen einen sogenannten Externen Identifier entgegen. Dieser identifiziert eindeutig eine Instanz eines Teils. Beispiel: Das Konzept »Sechs-kantschraube« hat in der Teiledatenbank genau eine gespeicherte Instanz, das ist eine Schraube mit definierten Werteeigenschaften.

Zur Abfrage stehen folgende Oracle-Prozeduren zur Verfügung:

GET_PROP_VALS_STRING Nimmt als IN-Parameter eine Externe Produkt-ID entgegen und liefert eine Tabelle vom Typ PROP_STRING_NTT zurück. Diese Tabelle beinhaltet die folgenden Werte:

IRDI Eindeutiger Identifier der Teileeigenschaft.

VALUE Wert der mittels IRDI identifizierten Teileeigenschaft.

UNIT Einheit der Teileeigenschaft.

PREFIX Prefix für den konkreten Wert.

TOLERANCE Wertetoleranzangabe.

VALUE_ID Identifier des konkreten Wertes.

GET_PROP_VALS_NUMBER Nimmt als IN-Parameter eine Externe Produkt-ID entgegen und liefert eine Tabelle vom Typ PROP_NUMBER_NTT zurück. Die Rückgabetafelle entspricht der Tabelle von PROP_STRING_NTT, bis auf die Typisierung des Values.

GET_PROP_VALS_REFERENCES Nimmt als IN-Parameter eine Externe Produkt-ID entgegen und liefert eine Tabelle vom Typ PROP_REF_NTT zurück. Diese Tabelle beinhaltet die folgenden Werte:

IRDI Eindeutiger Identifier der Teileeigenschaft.

VALUE Wert der mittels IRDI identifizierten Teileeigenschaft.

GET_PROP_VALS_LIST_NUMBER Nimmt als IN-Parameter eine Externe Produkt-ID entgegen und liefert eine Tabelle vom Typ PROP_LIST_NUMBER_NTT zurück. Stellt eine Liste von Zahlenwerten dar. Diese Tabelle beinhaltet die folgenden Werte:

IRDI Eindeutiger Identifier der Teileeigenschaft.

POS_IN_LIST Position des Wertes der Teileeigenschaft in der Liste.

VALUE Wert der mittels IRDI identifizierten Teileeigenschaft.

UNIT Einheit der Teileeigenschaft.

PREFIX Prefix für den konkreten Wert.

TOLERANCE Wertetoleranzangabe.

VALUE_ID Identifier des konkreten Wertes.

GET_PROP_VALS_LIST_STRING Nimmt als IN-Parameter eine Externe Produkt-ID entgegen und liefert eine Tabelle vom Typ PROP_LIST_STRING_NTT zurück. Stellt eine Liste von String-Werten dar. Diese Tabelle beinhaltet die folgenden Werte:

IRDI Eindeutiger Identifier der Teileeigenschaft.

POS_IN_LIST Position des Wertes der Teileigenschaft in der Liste.

VALUE Wert der mittels IRDI identifizierten Teileigenschaft.

UNIT Einheit der Teileigenschaft.

PREFIX Prefix für den konkreten Wert.

TOLERANCE Wertetoleranzangabe.

VALUE_ID Identifier des konkreten Wertes.

GET_PROP_VALS_MULTILIST_NUMBER Nimmt als IN-Parameter eine Externe Produkt-ID entgegen und liefert eine Tabelle vom Typ

PROP_MULTILIST_NUMBER_NTT

zurück. Stellt eine Liste von Zahlenwerten in mehreren Dimensionen dar (Matrix). Diese Tabelle beinhaltet die folgenden Werte:

IRDI Eindeutiger Identifier der Teileigenschaft.

POS_IN_LIST Position des Wertes der Teileigenschaft in der Liste.

LIST_COORD_T_REC Ein Eintrag vom Typ LIST_COORD_T_REC, gleichsam einem Feld in einer Matrix. Besteht aus folgenden Einträgen:

DIM Zeigt die Dimension an.

POS Zeigt die Position in der Dimension an.

VAL Wert der mittels IRDI identifizierten Teileigenschaft.

UNIT Einheit der Teileigenschaft.

PREFIX Prefix für den konkreten Wert.

TOLERANCE Wertetoleranzangabe.

VALUE_ID Identifier des konkreten Wertes.

GET_PROP_VALS_MULTILIST_STRING Nimmt als IN-Parameter eine Externe Produkt-ID entgegen und liefert eine Tabelle vom Typ

PROP_MULTILIST_NUMBER_NTT

zurück. Stellt eine Liste von String-Werten in mehreren Dimensionen dar (Matrix). Diese Tabelle beinhaltet die folgenden Werte:

IRDI Eindeutiger Identifier der Teileigenschaft.

POS_IN_LIST Position des Wertes der Teileigenschaft in der Liste.

LIST_COORD_T_REC Ein Eintrag vom Typ LIST_COORD_T_REC, gleichsam einem Feld in einer Matrix. Besteht aus folgenden Einträgen:

- DIM** Zeigt die Dimension an.
- POS** Zeigt die Position in der Dimension an.
- VAL** Wert der mittels IRDI identifizierten Teileeigenschaft.
- UNIT** Einheit der Teileeigenschaft.
- PREFIX** Prefix für den konkreten Wert.
- TOLERANCE** Wertetoleranzangabe.
- VALUE_ID** Identifier des konkreten Wertes.

GET_PROP_VALS Die globale Methode, welche alle Prozeduren aufruft. Nimmt als IN-Parameter eine Externe Produkt-ID entgegen und liefert die folgenden Tabellen zurück:

- PROP_NUMBER_NTT
- PROP_STRING_NTT
- PROP_REF_NTT
- PROP_LIST_NUMBER_NTT
- PROP_LIST_STRING_NTT
- PROP_MULTILIST_NUMBER_NTT
- PROP_MULTILIST_STRING_NTT

Diese Tabellen sind bereits einzeln oben beschrieben.

Für genaue konzeptuelle Beschreibungen sei auf die Abschlussarbeit von Herrn Karsten Mende verwiesen.

5.4.1. Problemstellung - Externer Identifier

Hier stellt sich das Problem, dass eine query.xsd generell immer auf die IRDI eines Konzeptes basiert.

Beispiel: »Gib mir bitte alle Instanzen und Werte der Eigenschaften des Klassenkonzeptes mit dem Identifier 0173-1#01-BAD803#2 (Skalpell)«

Der Query des Standards fragt folglich explizit nach den Instanzen eines Konzeptes (hier Skalpell). Die Prozeduren benötigen allerdings einen nicht im Standard definierten Identifier einer konkreten Instanz, um alle Eigenschaftswerte zu ermitteln. Es ist folglich nicht möglich, die Prozeduren direkt mittels IRDI aufzurufen und die entsprechend benötigten Rückgabedaten zu erhalten.

Lösung

Zwei Lösungsoptionen stellen sich hier zur Auswahl:

1. Anpassen der Datenbank-Prozeduren, sodass anstatt eines externen Identifiers eine IRDI entgegengenommen werden kann.
2. Separate Abfrage an die Datenbank in der Applikation realisieren. Diese Abfrage ermittelt anhand der vom Query übergebenen IRDI alle externen Identifier der Instanzen dieses Konzeptes. Anschließend können die Prozeduren mit den ermittelten externen Identifiern aufgerufen werden.

Es wurde die Lösung Nummer zwei gewählt. Hierfür gibt es mehrere Gründe.

Der Hauptgrund ist, dass zum Zeitpunkt der Erstellung dieser Arbeit, die Abschlussarbeit von Herrn Mende noch nicht abgeschlossen ist. Es ist damit nicht auszuschließen, dass sich Prozeduren oder Logik in den Prozeduren zu einem späteren Zeitpunkt noch ändern. Ein weiterer Grund für die separate Abfrage ist, dass dies in der Ebene der Applikationslogik für einen Nicht-Oracle-Datenbank-Experten einfacher zu implementieren, und damit weniger fehleranfällig ist. Die Prozeduren sind bereits sehr komplex und im Detail nicht einfach zu verstehen. Die separate Abfrage hingegen ist relativ trivial, wie die Lösung, das Listing 5.9, zeigt. Ferner stellt diese Abfrage eine Teifunktionalität dar und ist somit prima separat testbar.

```
1  SELECT o.DI_ID FROM DE_CLASS c, DO_OBJECT o WHERE c.ID = o.C_ID AND c.IRDI = ?
```

Listing 5.9: SQL Query - Externe IDs abfragen

Zur Erklärung des Listing 5.9: DI_ID ist die eindeutige ID einer Instanz, also eines konkreten Teils. Das Prädikat IRDI = ? nimmt die IRDI des eigentlichen Queries gemäß query.xsd entgegen. Das Ergebnis ist eine Liste der externen Identifier, welche dann je an die Prozedur übergeben werden können, um die Daten der konkreten Instanzen des Konzeptes nach IRDI zu erhalten.

Der Ablauf einer Abfrage in der Applikation zeigt das Sequenzdiagramm in Abbildung 5.1. Hier werden beispielhaft anhand eines Simple Queries dargestellt, welche Klassen involviert sind. Die Aufrufe Richtung Datenbank von der Klasse PlibDAO werden der Einfachheit halber nicht dargestellt.

5.4.2. Problemstellung - Daten der Teile-Eigenschaften

Die genannten Prozeduren liefern die Daten der Eigenschaften eines spezifizierten Teiles nach IRDI. Die Liste der zurückgegebenen Daten scheint auf den ersten Blick ausreichend, denn man erhält die folgenden Eigenschaften der Prozedur:

5. Implementierung

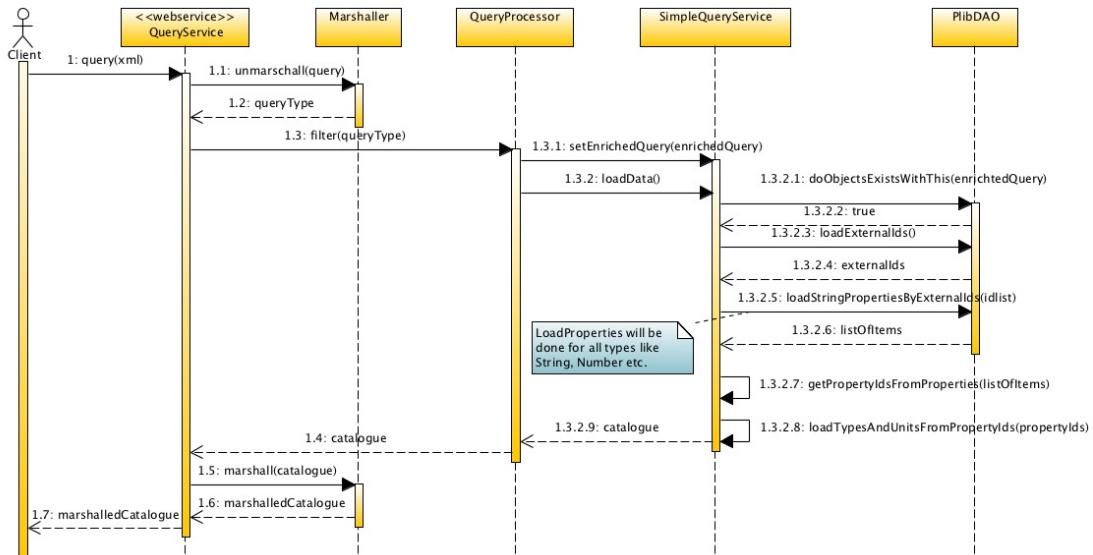


Abbildung 5.1.: Sequenzdiagramm Simple Query

Schritt	Beschreibung
1	Der Client sendet das Query-XML-File über den Webservice an den QueryService.
1.1	Der Query wird umgewandelt in ein Java Modell.
1.2	Das queryType Modell wird zurückgeliefert.
1.3	Die Filter-Methode des QueryProcessors wird aufgerufen, nach Erkennung, ob es sich um einen Simple Query oder Parametric Query handelt, wird der Query mit zusätzlichen Informationen angereichert.
1.3.1	Der angereicherte Query wird an den ausgewählten QueryService weitergeleitet.
1.3.2	loadData wird aufgerufen, um die Daten des Queries zu laden.
1.3.2.1	Es wird geprüft ob die angegebenen Objekte des Queries in der Datenbank verfügbar sind.
1.3.2.2	true wird zurückgeliefert.
1.3.2.3	Die external IDs der mittels IRDI übergebenen Objekte werden geladen. Hinweis: Wie bereits erwähnt, liefern die Prozeduren keine Möglichkeit eine IRDI unmittelbar zu übergeben, sondern nur den external Identifier des konkreten Objektes.
1.3.2.4	Die externen IDs der Teile werden zurückgeliefert.
1.3.2.5	Alle Eigenschaften der Teile werden geladen.
1.3.2.6	Die Teile mit den Eigenschaften werden zurückgeliefert.
1.3.2.7	Alle IRDIs der Eigenschaften müssen aus den Ergebnissen ausgelesen und in das Katalog-Modell übertragen werden.
1.3.2.8	Alle Typen und Einheiten müssen aus den Ergebnissen ausgelesen und in das Katalog-Modell übertragen werden.
1.3.2.9	Das gefüllte Katalog-Modell wird zurückgeliefert.
1.4	Das gefüllte Katalog-Modell wird zurückgeliefert.
1.5	Das gefüllte Katalog-Modell wird in XML-Repräsentation umgewandelt.
1.6	Die XML-Repräsentation des Katalog-Modells wird zurückgeliefert.
1.7	Die XML-Repräsentation des Katalog-Modells wird an den Client zurückgeliefert.

Tabelle 5.1.: Erläuterung des Abfrageablaufes der Applikation

GET_PROP_VALS_STRING

- IRDI
- UNIT
- TOLERANCE
- VALUE
- PREFIX
- VALUE_ID

»VALUE« ist der Wert der Eigenschaft, »UNIT« die Einheit. Ein genauer Blick auf das Schema des Standards (siehe Listing I.5 aus Anhang I) zeigt auf, dass es konkrete Typen für die Einheit des Wertes gibt, z.B. boolean_value, complex_value, composite_value, controlled_value usw. Für eine komplette Liste siehe Listing I.5. Die Daten aus dem Response der Prozedur bezüglich UNIT, PREFIX und TOLERANCE kommen aus der Relation DO_ADDITIONAL_DATA, die mit der jeweiligen Datenbanktabelle der Eigenschaftswerte, das sind beispielsweise DO_STRING, DO_NUMBER, DO_REFERENCES usw., verknüpft wird. Das bedeutet, dass einer Eigenschaft einer Instanz eines Teiles beispielsweise die Einheit (UNIT) gesetzt werden kann. Dies erscheint an dieser Stelle fragwürdig, da es bedeuten würde, für jede Instanz jeweils eine Einheit zu speichern. Es stellt sich hier das Problem, dass anhand des String-Wertes aus UNIT nicht klar ist, auf welchen Typen aus value.xsd gemappt werden soll. Es liegt ein Heterogenitätskonflikt der beiden Datenschemata vor. Ferner wurde festgestellt, dass die Metadaten, nämlich der Relation DE_PROPERTY über DE_DATA_TYPE auf DE_UNIT und DE_PREFIX verwiesen wird, womit allen Instanzen dieser Eigenschaft die Einheit (UNIT) und der Prefix (PREFIX) gesetzt werden kann. Dies erscheint sinnvoller und würde auch ein Mapping ermöglichen.

Lösung

In einem Gespräch mit Karsten Mende wurde diese Situation besprochen. Herr Mende erläuterte, dass der Standard ISO 13584-42 (vgl. ISO09a) die Möglichkeit gibt, die Einheit der konkreten Instanzen von Eigenschaften zu überschreiben. Auf Grund der Heterogenität und Repräsentation in Strings empfiehlt Herr Mende an dieser Stelle auf die Metadatenbeschreibungen von UNIT, PREFIX und TOLERANCE der Relationen DE_PROPERTY, DE_DATA_TYPE, DE_UNIT und DE_PREFIX zurückzugreifen. Da dies sinnvoll erscheint und ferner die Implementierung vereinfacht, wurde dies in der Form umgesetzt. Es bleibt an dieser Stelle zu bemerken, dass für eine spätere Implementierung und Unterstützung dieser Funktionalität zu analyseren ist, wie die Heterogenität zu lösen ist. Ggf. muss dazu eine Mappingtabelle erstellt werden.

Fazit: Zur Zeit werden folglich gesetzte Werte von UNIT, PREFIX und TOLERANCE in DO_ADDITIONAL_DATA ignoriert und nur die Metadatenrepräsentationen benutzt.

5.4.3. Problemstellung - Aufruf der Prozeduren mit Java

Um die Prozeduren von Oracle aufzurufen wurde Spring Data Oracle eingesetzt.

5. Implementierung

Die eigentlichen Hilfsprozeduren, welche von Karsten Mende zur Verfügung gestellt werden, liefern keine Rückgabewerte. Das Listing 5.10 zeigt die erste Testimplementierung⁹.

Der Grund, weshalb keine Rückgabewerte geliefert werden, ist, dass die Hilfsprozeduren Ausgaben mit der Oracle-Funktion »dbms_output.put_line« erzeugen. Auf diese manuell erzeugten Ausgabetexte kann nicht zugegriffen werden.

```
1  /**
2   * This class calls the HILF_GET_PROP_VALS procedure.
3   * However, during tests I found out that we have no real out-parameter.
4   * Thus either must be changed or Queries from the procedures must be used as
5   * simple sql.
6   */
7  public class PropertyValuesStoredProcedure extends StoredProcedure {
8      private static final String PROCEDURE_NAME = "PACK_PROPERTY.
9          HILF_GET_PROP_VALS_STRING";
10
11     /**
12      * Logger instance
13      */
14     private static Logger LOGGER = Logger.getLogger(TestProcedure.class);
15
16     public PropertyValuesStoredProcedure(DataSource ds) {
17         super(ds, PROCEDURE_NAME);
18         declareParameter(new SqlParameter("OBJ_ID", Types.VARCHAR));
19         compile();
20     }
21
22     public String execute(String objectId) {
23         Map in = new HashMap();
24         in.put("OBJ_ID", objectId);
25         Map out = execute(in);
26         LOGGER.info("Output: " + out);
27         if (!out.isEmpty())
28             return out.get("OBJ_ID").toString();
29         else
30             return null;
31     }
32 }
```

Listing 5.10: Spring Data Oracle - Klasse zum Aufruf der Hilfsprozeduren

Im Listing 5.10 leitet die Klasse `PropertyValuesStoredProcedure` von der Spring-Klasse `StoredProcedure` ab. Im Konstruktor werden die Parameter definiert, in diesem Falle lediglich `OBJ_ID` als Externer Identifier. Ferner wird die Methode `execute` überschrieben, welche den Aufruf der Prozedur kapselt.

Listing 5.11 zeigt eine Testklasse, welche die Prozedur auruft. Die Testklasse verfügt über zwei Testmethoden:

testShowsThatWeHaveThatinDB Diese Methode über gibt einen gültigen und in der Datenbank vorhandenen Externen Identifier. Wird die Hilfsmethode über SQL di-

⁹Das Listing sowie alle nachfolgende Listings verzichten auf Import- und Package-Anweisungen um die Lesbarkeit zu erhöhen. Alle Quelltexte liegen der Arbeit bei.

rekt auf der Datenbank mit diesem Parameter aufgerufen, so wird Folgendes ausgegeben:

```
LIST_STRING_PROPS 1. Stelle Property-IRDI : val : Euro
LIST_STRING_PROPS 2. Stelle Property-IRDI : val : 2008-12-31
LIST_STRING_PROPS 3. Stelle Property-IRDI : val : 170202
LIST_STRING_PROPS 4. Stelle Property-IRDI : val : 24-31-01-01
LIST_STRING_PROPS 5. Stelle Property-IRDI : val : Rundolf-75
LIST_STRING_PROPS 6. Stelle Property-IRDI : val : 2008-05-01
```

Es ist eine Liste mit sechs Eigenschaftswerten. Wird die Prozedur über den Test aufgerufen, so ist der Testablauf zwar erfolgreich, allerdings liefert dieser kein Ergebnis zurück. Ein solches Verhalten wurde nicht erwartet.

testShouldThrowExceptionAsExtIDIsNotInDB Diese Testmethode nutzt einen nicht vorhandenen Externen Identifier. Via SQL direkt auf der Datenbank ausgeführt, liefert die Testmethode: »No Data found« zurück. Wird die Prozedur über den Test aufgerufen, so wird eine Exception geworfen, welche ebenfalls den Text »No Data found« beinhaltet. Dies ist somit ein korrektes Verhalten.

Der Grund, weshalb der Aufruf der Prozedur mit einem gültigen externen IRDI keinen Wert zurückliefert, ist der, dass die Hilfsmethoden mit dem Prefix

HILF_*(z.B. HILF_GET_PROP_VAL_STRING)

in der Prozedurdefinition Ausgaben mittels

DBMS_OUTPUT.PUT_LINE

erzeugen. Das ist eine Oracle spezifische Funktion um Ausgaben zu erzeugen. Hierauf kann nicht zugegriffen werden. Ferner sind die Daten nicht strukturiert.

```
1  @RunWith(SpringJUnit4ClassRunner.class)
2  @ContextConfiguration(locations = {"beans.xml"})
3  public class PropertyValuesStoredProcedureIT {
4      @Autowired
5      javax.sql.DataSource ds;
6
7      @Test
8      public void testShowsThatWeHaveThatInDB() {
9          PropertyValuesStoredProcedure propertyValuesStoredProcedure = new
10             PropertyValuesStoredProcedure(ds);
11             propertyValuesStoredProcedure.execute("EXT_50766884");
12     }
13
14     @Test(expected = RuntimeException.class)
15     public void testShouldThrowExceptionAsExtIDIsNotInDB() {
16         PropertyValuesStoredProcedure propertyValuesStoredProcedure = new
17             PropertyValuesStoredProcedure(ds);
18             propertyValuesStoredProcedure.execute("EXT_555");
19     }
20 }
```

5. Implementierung

```
19     public DataSource getDs() {
20         return ds;
21     }
22
23     public void setDs(DataSource ds) {
24         this.ds = ds;
25     }
26
27 }
```

Listing 5.11: Spring Data Oracle - Testklasse zum Aufruf der Hilfsprozeduren

Lösung

Der nächste Test muss demnach eine Ebene tiefer ansetzen, und zwar dort, wo die Daten in Typstrukturen genutzt werden. Das sind die konkreten Prozeduren wie z.B. GET_PROP_VALS_STRING. Diese liefern Tabellen vom Typ konkreter Rückgabetypen wie in Abschnitt 5.4 beschrieben.

Das Listing 5.12 zeigt beispielhaft die Zugriffsklasse für die Prozedur

GET_PROP_VALS_STRING.

```
1
2 public class TestProcedure extends StoredProcedure {
3
4     private static final String SQL = "PACK_PROPERTY.GET_PROP_VALS_STRING";
5
6     /**
7      * Logger instance
8      */
9     private static Logger LOGGER = Logger.getLogger(TestProcedure.class);
10
11    public TestProcedure(DataSource ds) {
12        super(ds, SQL);
13        declareParameter(new SqlParameter("EXT_PROD_ID", OracleTypes.VARCHAR));
14
15        declareParameter(new SqlOutParameter("OUTTABLE_STRING", OracleTypes.STRUCT, "PACK_PROPERTY.PROP_STRING_NTT", new SqlReturnSqlData(PropStringNtt.class)));
16        compile();
17    }
18
19    public String execute(Long propertyId, Long is_cvp) {
20        Map in = new HashMap();
21        in.put("EXT_PROD_ID", propertyId);
22        in.put("is_cvp", is_cvp);
23        Map out = execute(in);
24        if (!out.isEmpty())
25            return out.get("message").toString();
26        else
27            return null;
28    }
29
30    public String execute(String propertyId) {
31        Map in = new HashMap();
32        in.put("EXT_PROD_ID", propertyId);
```

```

33     Map<String, Object> out = execute(in);
34     if (!out.isEmpty())
35         return out.get("OUTTABLE_STRING").toString();
36     else
37         return null;
38 }
39 }
```

Listing 5.12: Spring Data Oracle - Aufruf der Prozeduren mit Rückgabebäbeln

5.4.4. Problemstellung - RECORD-Types als Argumente werden nicht von Java-JDBC unterstützt

Im Listing 5.12 wird ein Test dargestellt, der vermeintlich die Prozedur

»GET_PROP_VALS_STRING«

aufrufen kann. Dies ist der nächste logische Schritt, welcher allerdings nicht funktioniert. Beim Aufruf der Prozedur von Java wird eine Fehlermeldung zurückgegeben, die besagt, dass der Rückgabetypr
PACK_PROPERTY.PROP_STRING_NTT
nicht erkannt wurde.

Zur Erklärung dazu, PACK_PROPERTY ist das von Herrn Mende definierte Paket, welches zum Ziel hat, zusammengehörige Prozeduren, Typen und Funktionen zu gruppieren. Die Erstellung von Paketen erhöht die Wart- und Lesbarkeit des Oracle-PL/SQL Codes.

Eine Recherche hat ergeben, dass eine Oracle Prozedur welche RECORD-Types als IN oder OUT-Parameter benutzt, mittels Java JDBC-Treiber nicht aufgerufen werden kann.

Der SQLJ Developer's Guide and Reference sagt dazu (Kap. 5 Wri02):

»Oracle SQLJ and JDBC do not support calling arguments or return values of the PL/SQL BOOLEAN type or RECORD types. Also, when using the Thin driver, they do not support calling arguments or return values of PL/SQL TABLE types (known as indexed-by tables). TABLE types are supported for the OCI driver, however.«

Lösung

Es gibt mehrere Lösungsansätze. Der erste Ansatz ist autark zu lösen, ohne Anpassung der Prozeduren. Hierzu werden nicht die Prozeduren eingesetzt, sondern die benutzten SQL-Anfragen aus den Prozeduren werden eigenständig benutzt. Der Vorteil dieser Lösung ist, dass die Prozeduren nicht angepasst werden müssen. Der Nachteil ist allerdings, dass die Prozeduren, welche in der Arbeit von Herrn Mende entwickelt wurden,

5. Implementierung

in der Form gar nicht mehr genutzt werden. Das ist nicht das eigentliche Ziel, führt trotzdem zum Ergebnis.

Die erste Lösung wurde zu Testzwecken implementiert, siehe Listing 5.13.

```
1  SELECT
2      P.ID ,
3      P.IRDI ,
4      LOJ.VALUE ,
5      LOJ.UNIT ,
6      LOJ.PREFIX ,
7      LOJ.TOLERANCE ,
8      LOJ.VALUE_ID
9  FROM
10     (SELECT
11         DI_ID ,
12         P_ID ,
13         VALUE ,
14         UNIT ,
15         PREFIX ,
16         TOLERANCE ,
17         VALUE_ID
18     FROM
19         DO_STRING
20     LEFT OUTER JOIN
21     (SELECT
22         DI_ID ,
23         P_ID ,
24         UNIT ,
25         PREFIX ,
26         TOLERANCE ,
27         VALUE_ID
28     FROM
29         DO_ADDITIONAL_DATA
30     WHERE
31         ERR_CODE = 1)
32     USING (DI_ID , P_ID)
33             WHERE (DI_ID =?)
34                     AND (DO_STRING.ERR_CODE = 1))
35     LOJ JOIN DE_PROPERTY P
36     ON LOJ.P_ID = P.ID
```

Listing 5.13: SELECT-Abfrage für String-Properties aus den Prozeduren

Wenngleich die Abfrage zum Ergebnis führt, so ist sie sehr komplex. Daher sollte wie geplant die Abfragekomplexität in der Prozedur gekapselt sein. Damit sind wir gleich beim zweiten Lösungsansatz. Hierbei müssen die Prozeduren angepasst und als Rückgabetyp ein OBJECT-Typ benutzt werden.

Erst wurde die erste Lösung implementiert, um eine Basis für die weitere Entwicklung zu bilden, denn für die zweite Lösung muss Herr Mende die Prozeduren anpassen. So konnte die Entwicklung fortgeführt werden und gleichzeitig Herr Mende die Änderungen für die zweite, gleichsam die empfohlene Lösung entwickeln. Listing 5.14 zeigt den von Herrn Mende neuen Beispieltypen PROP_STRING_OBJ. Analog zu den anderen Typen aus Abschnitt 5.4 gibt es folglich weitere Typdefintionen wie

z.B. PROP_NUMBER_OBJ oder PROP_REFERENZ_OBJ. Die Prozeduren mit den RECORD Typen wurden beibehalten und neue Prozeduren entwickelt. Die Benamung ist GET_OBJ_STRING, GET_OBJ_NUMBER usw.

```

1  create or replace
2    TYPE      "PROP_STRING_OBJ_T" AS OBJECT(
3      P_IRDI    VARCHAR2(4000),
4      "VALUE"   VARCHAR2(4000),
5      unit      VARCHAR2(30),
6      prefix    VARCHAR2(30),
7      tolerance NUMBER,
8      VALUE_ID  NUMBER
9  );
10
11 create or replace
12 TYPE    "PROP_STRING_OBJ_NTT"   AS TABLE OF  "PROP_STRING_OBJ_T";

```

Listing 5.14: PROP_STRING_OBJ Typen

5.4.5. Problemstellung - Fehlender Fremdschlüsselidentifier in Prozedurrückgabedaten

Wie bereits in Unterabschnitt 5.4.2 beschrieben, liefern die Prozeduren folgenden Attribute pro Teil:

IRDI, VALUE, UNIT, PREFIX, TOLERANCE, VALUE_ID

Ferner wird aufgezeigt, dass die Tabelle DE_PROPERTY über DE_DATA_TYPE auf DE_UNIT und DE_PREFIX an alle relevanten Daten für das semantische Mapping kommen. Es stellt sich aber das Problem, dass die Rückgabedaten keinen Identifier beinhalten (P_ID - Property ID), über welche die Tabellen miteinander verknüpft werden können (Join).

Lösung

Es gibt zwei Lösungsansätze. Die erste Lösung ist, weitere SQL Anfragen zu stellen, um die benötigten Daten zu erfragen. Nachteilig ist an dieser Lösung, dass bereits für die Ermittlung des Externen Identifiers aus Unterabschnitt 5.4.1 zusätzliche Anfragen nötig sind. In diesem Fall kommen weitere hinzu.

Die andere Lösung ist die Anpassung der Prozeduren und Rückgabetypen, sodass unmittelbar die Property ID mitgeliefert wird. Das hat den Vorteil, dass keine weitere SQL-Anfrage an die Datenbank gesendet werden muss. Innerhalb der Prozedur wird ein weiteres Attribut selektiert.

Es wurde die zweite Lösung gewählt. Hierbei müssen alle Typdefinitionen angepasst werden, siehe beispielhaft Listing 5.15. Dazu wird die Property ID eingefügt:

»P_ID NUMBER«.

5. Implementierung

```
1 create or replace
2  TYPE      "PROP_STRING_OBJ_T" AS OBJECT(
3    P_ID      NUMBER,
4    P_IRDI    VARCHAR2(4000),
5    "VALUE"   VARCHAR2(4000),
6    unit      VARCHAR2(30),
7    prefix    VARCHAR2(30),
8    tolerance NUMBER,
9    VALUE_ID  NUMBER
10 );
11
12 create or replace
13 TYPE    "PROP_STRING_OBJ_NTT" AS TABLE OF "PROP_STRING_OBJ_T";
```

Listing 5.15: PROP_STRING_OBJ Typanpassung

Das Listing 5.16 zeigt die gesamte GET_OBJ_STRING Prozedur samt Anpassung.

```
1 PROCEDURE GET_OBJ_STRING (
2     EXT_PROD_ID           IN VARCHAR2,
3     OUTTBL_OBJ_STRING    OUT PROP_STRING_OBJ_NTT,
4     CREATED_VIEW          IN NUMBER := NULL
5 )
6 IS
7     CURSOR CUR_DO_STRING (IN_OBJ_ID NUMBER)
8     IS
9         SELECT p.id, P.IRDI, LOJ.VALUE, LOJ.UNIT, LOJ.PREFIX, LOJ.TOLERANCE, LOJ.
10            VALUE_ID
11        FROM (SELECT DI_ID, P_ID,  VALUE, UNIT, PREFIX, TOLERANCE, VALUE_ID
12                  FROM DO_STRING LEFT OUTER JOIN (SELECT DI_ID, P_ID, UNIT, PREFIX,
13                                              TOLERANCE, VALUE_ID
14                                FROM DO_ADDITIONAL_DATA
15                               WHERE ERR_CODE = 1)
16                    USING (DI_ID, P_ID)
17                   WHERE (DI_ID =IN_OBJ_ID) AND (DO_STRING.ERR_CODE = 1))LOJ JOIN
18                     DE_PROPERTY P
19                   ON LOJ.P_ID = P.ID
20                   ORDER BY P.IRDI;
21
22     OBJ_ID_NR             NUMBER;
23     OUTTABLE_STRING        PROP_STRING_NTT := PROP_STRING_NTT();
24     OUTTBL_OBJ_STRING_ELEM PROP_STRING_OBJ_T := PROP_STRING_OBJ_T(NULL, NULL, NULL,
25                           NULL, NULL, NULL);
26     I                      INTEGER;
27     EMPTY_OUTTBL_OBJ_STRING PROP_STRING_OBJ_NTT := PROP_STRING_OBJ_NTT();
28
29 BEGIN
30     OUTTBL_OBJ_STRING := EMPTY_OUTTBL_OBJ_STRING;
31
32     IF CREATED_VIEW IS NULL THEN
33         EXECUTE IMMEDIATE 'SELECT DI_ID FROM DO_OBJECT WHERE EXT_PROD_ID = ''' ||
34                           EXT_PROD_ID || ''' AND CREATED_VIEW IS NULL' INTO OBJ_ID_NR;
35     ELSE
36         EXECUTE IMMEDIATE 'SELECT DI_ID FROM DO_OBJECT WHERE EXT_PROD_ID = ''' ||
37                           EXT_PROD_ID || ''' AND CREATED_VIEW = ' || CREATED_VIEW INTO OBJ_ID_NR ;
38     END IF;
39
40     OPEN CUR_DO_STRING(OBJ_ID_NR);
41     FETCH CUR_DO_STRING BULK COLLECT INTO OUTTABLE_STRING;
42     CLOSE CUR_DO_STRING;
```

```

38  IF OUTTABLE_STRING.FIRST IS NULL THEN
39      DBMS_OUTPUT.PUT_LINE('OUTTABLE_STRING ist leer!');
40  ELSE
41      DBMS_OUTPUT.PUT_LINE('OUTTABLE_STRING ist NICHT leer!');
42      FOR I IN OUTTABLE_STRING.FIRST .. OUTTABLE_STRING.LAST LOOP
43
44          DBMS_OUTPUT.PUT_LINE('OUTTABLE_STRING(I).PROP_IRDI: '|| OUTTABLE_STRING(I).
45              PROP_IRDI);
46          OUTTBL_OBJ_STRING_ELEM.P_ID := OUTTABLE_STRING(I).PROP_ID;
47          OUTTBL_OBJ_STRING_ELEM.P_IRDI := OUTTABLE_STRING(I).PROP_IRDI;
48          OUTTBL_OBJ_STRING_ELEM.VALUE := OUTTABLE_STRING(I).VAL;
49          OUTTBL_OBJ_STRING_ELEM.UNIT := OUTTABLE_STRING(I).UNIT;
50          OUTTBL_OBJ_STRING_ELEM.PREFIX := OUTTABLE_STRING(I).PREFIX;
51          OUTTBL_OBJ_STRING_ELEM.TOLERANCE := OUTTABLE_STRING(I).TOLERANCE;
52          OUTTBL_OBJ_STRING_ELEM.VALUE_ID := OUTTABLE_STRING(I).VALUE_ID;
53          OUTTBL_OBJ_STRING.EXTEND;
54          OUTTBL_OBJ_STRING(OUTTBL_OBJ_STRING.LAST) := OUTTBL_OBJ_STRING_ELEM;
55          DBMS_OUTPUT.PUT_LINE('Wertzuweisung an OUTTBL_OBJ_STRING erfolgreich!');
56      END LOOP;
57  END IF;
58 END GET_OBJ_STRING;

```

Listing 5.16: GET_OBJ_STRING Anpassung auf Property ID

Es müssen drei Stellen angepasst werden:

- Das SELECT Statement für die Rückgabe, sodass die Property ID übergeben wird:

vorher SELECT

```
P.IRDI, LOJ.VALUE, LOJ.UNIT, LOJ.PREFIX,
LOJ.TOLERANCE, LOJ.VALUE_ID
```

...

nachher SELECT

```
P.ID, P.IRDI, LOJ.VALUE, LOJ.UNIT, LOJ.PREFIX,
LOJ.TOLERANCE, LOJ.VALUE_ID
```

...

- Die Erweiterung bei Instanzierung des Tabelleneintrages für den Tabellentypen um ein Attribut (NULL hinzugefügt):

vorher OUTTBL_OBJ_STRING_ELEM PROP_STRING_OBJ_T :=
PROP_STRING_OBJ_T(NULL, NULL, NULL, NULL, NULL, NULL);

nachher OUTTBL_OBJ_STRING_ELEM PROP_STRING_OBJ_T :=
PROP_STRING_OBJ_T(NULL, NULL, NULL, NULL, NULL, NULL, NULL);

- Hinzufügen des Mappings von der RECORD-Tabelle auf den OBJECT-Typen:

vorher -

nachher OUTTBL_OBJ_STRING_ELEM.P_ID :=
OUTTABLE_STRING(I).PROP_ID;

5.5. Fehlerbehandlung

Treten während der Verarbeitung Fehler auf, wie z.B. ein Validierungsfehler des übergebenen XML Queries, so ergibt sich das Problem, dass diese Fehlermeldung nicht an den Klienten zurückgegeben werden kann. Das ist nicht möglich, da das aktuelle Schema der Rückgabedatei »query.xsd« (siehe Abschnitt I.1) kein Modell für Fehlernachrichten beinhaltet. Dieses Verhalten ist korrekt, denn hier wird ein Datenmodell beschrieben und Kontext- und Zusatzinformationen gehören nicht in dieses Modell, sondern sollten über die darüber liegende Ebene verarbeitet werden. Dies ist in der Lösung mit RESTful Webservice nicht sinnvoll, denn hier müsste das Protokoll, HTTP, benutzt werden um Fehler anzuzeigen. Da SOAP ein Protokoll ist, verfügen SOAP-Webservices per se über einen Fehlermechanismus genannt SOAP-Fault¹⁰.

Es bieten sich drei Möglichkeiten an. Die erste wäre beim Auftreten eines Fehlers in der Verarbeitung immer einen leeren Katalog zurückzuliefern. Der Vorteil dieser Lösung ist die Einfachheit. Der Nachteil ist, dass nach wie vor keinerlei Fehlerinformation an den Klienten zurückgeliefert wird.

Die zweite Möglichkeit ist ein weiteres Schema zu erstellen und dieses im Katalog-Schema zu referenzieren. In diesem »Fehler-Schema« wird dann ein FehlerTyp definiert, welcher Fehlernachrichten aufnehmen kann. Der Vorteil dieser Lösung ist, dass es einfach umzusetzen und in das vorhandenen Katalog-Schema zu integrieren ist. Diese Lösung wäre sowohl für RESTful Webservices als auch für SOAP-Webservices nutzbar. Der Nachteil ist, dass dies eine Anpassung des Katalog-Schemas bedeutet.

Die dritte Option ist es die Fehlerbehandlung über HTTP vorzunehmen. HTTP verfügt über Möglichkeiten Fehler anhand von Fehlercodes anzuzeigen. Um anhand der Antwort einen Fehler überhaupt erkennen zu können, wurde die zweite Lösung gewählt. Ferner ist diese Lösung unabhängig von der eingesetzten Webservice-Technologie. Das Listing 5.17 zeigt die Schemadatei error.xsd, welche einen Typen definiert (errorType), der zwei Fehlertexte beinhalten kann, shortErrorMessageType und longErrorMessageType.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns:err="error"
   targetNamespace="error" elementFormDefault="qualified" attributeFormDefault="unqualified">
3   <xss:complexType name="errorType">
4     <xss:sequence>
5       <xss:element name="shortErrorMessage" type="err:shortErrorMessageType" />
6       <xss:element name="longErrorMessage" type="err:longErrorMessageType" />
7     </xss:sequence>
8   </xss:complexType>
9   <xss:simpleType name="shortErrorMessageType">
10    <xss:restriction base="xss:string"/>
11  </xss:simpleType>
12  <xss:simpleType name="longErrorMessageType">
13    <xss:restriction base="xss:string"/>
```

¹⁰Siehe <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>

```

14  </xs:simpleType>
15  </xs:schema>
```

Listing 5.17: Fehlerbehandlung - Error Schemadatei

Diese lässt sich in die Katalog-Schemadatei einbinden mittels:

```
<xs:import namespace="error" schemaLocation="error.xsd" />
```

Anschließend wird ein globales Error-Element angegeben:

```
<xs:element name="error" type="err:errorType" />
```

Schließlich wird der Catalogue-Type neben dem Item-Type noch durch den Error Type erweitert, das zeigt Listing 5.18.

```

1  <xs:complexType name="catalogue_Type">
2    <xs:sequence>
3      <xs:element ref="cat:item" minOccurs="0" maxOccurs="unbounded"/>
4      <xs:element ref="cat:error" minOccurs="0" maxOccurs="1"/>
5    </xs:sequence>
6  </xs:complexType>
```

Listing 5.18: Fehlerbehandlung - Catalogue Type erweitern

Das Listing 5.19 zeigt eine Beispielantwort einer Fehlermeldung¹¹. Hier wurde eine ungültige IRDI im Query gesendet. Die shortErrorMessage besagt, dass es sich um einen Fehler beim Unmarshalling handelt. Der gesamte Java-Stacktrace aus der aufgetretenen Exception im Programm wird in der longErrorMessage angegeben. Hier kann genau nach der Ursache des Fehlers geforscht werden. In diesem Fall findet man folgende Fehlermeldung:

```
»Caused by: org.xml.sax.SAXParseException: cvc-pattern-valid:
Value '01x73x-1#01-AAA352#4' is not facet-valid with respect to pattern ... for type 'IRDI_type'.«
```

Es ist ein Fehler bei der Validierung der IRDI aufgetreten. Diese entspricht nicht dem erlaubten Muster.

```

1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <catalogue xmlns:ns2="urn:iso:std:iso:ts:29002:-4:ed-1:tech:xml-schema:basic"
3    xmlns="urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:catalogue" xmlns:ns4
4    ="error" xmlns:ns3="urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:value">
5    <error>
6      <ns4:shortErrorMessage>Error occurred during unmarshalling of XML</ns4:
7        shortErrorMessage>
8      <ns4:longErrorMessage>java.lang.RuntimeException: Error during
9        unmarshalling of XML.
10       at de.feuplib.xml.XMLMarshallerImpl.unmarshallXML(XMLMarshallerImpl.java:63)
11       at de.feuplib.webservice.rest.QueryRESTService.unmarshall(QueryRESTService.java
12         :151)
13       at de.feuplib.webservice.rest.QueryRESTService.query(QueryRESTService.java:91)
14       ...
15       at java.lang.Thread.run(Thread.java:695)
```

¹¹Das Beispiel bildet einen gekürzten Stacktrace ab, um nur die relevanten Teile aufzuzeigen.

```
11 Caused by: org.xml.sax.SAXParseException: cvc-pattern-valid: Value '01x73x-1#01-
12 AAA352#4' is not facet-valid with respect to pattern '[0-9]{4}-[0-9A-Z:_-
13 \.){1,35}(-[0-9A-Z:_\.){1,35}(-[0-9A-Z]{1,1}(-[0-9A-Z:_\.){1,70}))?)?)?(&#[0-9A-
14 Z]{2,2}-[0-9A-Z:_\.){1,131}#[0-9]{1,10})?|[0-9]{4}-[0-9A-Z:_\.){1,35}(-[0-9A-Z
15 :_\.){1,35})?--[0-9A-Z:_\.){1,70}(&#[0-9A-Z]{2,2}-[0-9A-Z:_-
16 \.){1,131}#[0-9]{1,10})?|[0-9]{4}-[0-9A-Z:_\.){1,35}---[0-9A-Z:_-
17 \.){1,70}(&#[0-9A-Z]{2,2}-[0-9A-Z:_\.){1,131}#[0-9]{1,10}))?' for type '
18 IRDI_type'.
12 at com.sun.org.apache.xerces.internal.util.ErrorHandlerWrapper.
13     createSAXParseException(ErrorHandlerWrapper.java:195)
14 ...
15 ...
16 ... 38 more
17 &lt;/error&gt;
18 &lt;/catalogue&gt;</pre></div><div data-bbox="178 311 761 329" data-label="Caption"><p><b>Listing 5.19:</b> Fehlerbehandlung - Beispielantwort mit Validierungsfehler</p></div><div data-bbox="113 384 368 402" data-label="Section-Header"><h2>5.6. SOAP Webservice</h2></div><div data-bbox="113 432 827 470" data-label="Text"><p>Um beide Technologien miteinander zu vergleichen, wurde nach der RESTful Webser-<br/>vice-Implementierung noch ein SOAP-Webservice implementiert.</p></div><div data-bbox="113 483 828 579" data-label="Text"><p>Das Ergebnis ist, dass basierend auf der vorhandenen Implementierung ein weiterer<br/>Webservice in einer anderen Technologie mit recht geringem Aufwand erstellt werden<br/>kann. Im Rahmen der RESTful Webservice-Implementierung wurden bereits alle vorbe-<br/>reitenden Maßnahmen, wie Modellgenerierung aus den Schemadateien und Transfor-<br/>mation für die Abfrageprozeduren, getroffen.</p></div><div data-bbox="113 592 827 629" data-label="Text"><p>Für mehr Informationen zur Implementierung und den Beispielquellcode des SOAP-<br/>Webservices siehe Anhang G.</p></div><div data-bbox="113 671 696 692" data-label="Section-Header"><h2>5.7. Umfang und Abgrenzung der Implementierung</h2></div><div data-bbox="113 720 828 795" data-label="Text"><p>Der Umfang der Implementierung, gerade im Bereich der Transformation ist sehr<br/>groß. Ferner liegen der PLIB sehr wenige Testdaten vor. Daher werden für den Pro-<br/>totyp nur einige Prozeduren beispielhaft aufgerufen, und zwar GET_OBJ_STRING und<br/>GET_OBJ_NUMBER.</p></div><div data-bbox="113 809 544 827" data-label="Text"><p>Die folgenden Prozeduren werden nicht aufgerufen:</p></div><div data-bbox="142 845 803 917" data-label="List-Group"><ul style="list-style-type: none;"><li>• GET_OBJ_REF</li><li>• GET_OBJ_LIST_NUMBER</li><li>• GET_OBJ_LIST_STRING</li><li>• GET_OBJ_MULTILIST_NUMBER</li><li>• GET_OBJ_MULTILIST_STRING</li></ul></div><div data-bbox="113 956 140 973" data-label="Page-Footer"><hr/><p>52</p></div>
```

Das bedeutet, dass falls Daten in der PLIB in den passenden Tabellen zu den passenden Typen existieren, diese von der Abfrage nicht beachtet werden. Um die Implementierung folglich praxistauglich zu machen, müssen die Transformationen der Rückgabe der fehlenden Prozeduren vervollständigt werden.

Für die Transformation und das Füllen der Rückgabekatalogdaten wurden einige Parametric-Query-Abfragen nicht betrachtet. Das sind »DataCardinality«, »DataEnvironment« und »Subset«. Für diese Abfragetypen fehlen, wie auch oben erwähnt, Testdaten in der PLIB-Datenbank. Die entsprechenden Szenarien müssen aufwändig erstellt werden. Für einen möglichen Praxiseinsatz müsste die Implementierung vervollständigt werden. Zu Forschungs- und Demonstrationszwecken sind die implementierten Funktionalitäten ausreichend.

5.8. Zusammenfassung des Kapitels

Zunächst wird das Configuration Management erläutert, dazu gehört die Auswahl der Werkzeuge, Prozesse, Build-Umgebung, Frameworks und Plattformen. Es wird kurz auf das Thema Testen eingegangen und anhand einiger Beispiele erläutert. Da ein RESTful Webservice entwickelt wird, ist eine Beschreibung dieses Webservices Bestandteil des Kapitels. Es wird aufgezeigt, wie ein RESTful Webservice mit Hilfe des Frameworks Jersey entwickelt wird. Der nächste Schritt ist die Betrachtung des Datenmodells. Die Anfrage und die Antwort des Webservices ist mittels Schema beschrieben. Die Herausforderung ist die Umwandlung der XML-Daten gemäß Schema in ein Modell der Programmiersprache zu Weiterverarbeitung. Der wichtigste Punkt ist die Abfrage der Oracle PLIB-Prozeduren. Hier ergeben sich diverse Problemstellungen, wie semantische Heterogenität, fehlende Daten bei Eingabe- und Ausgabeschnittstelle der Prozeduren, als auch technische Schwierigkeiten, wie die fehlende Unterstützung von RECORD-Typen bei Abfrage mit Java JDBC.

Es wurde festgestellt, dass bei der Implementierung und Integration von Schnittstellen mit Problemen zu rechnen ist. Die Prozeduren wurden nicht mit Augenmerk auf die ISO 29002-31 entwickelt und weisen dadurch eine deutliche Heterogenität zu den tatsächlich benötigten Daten auf. Die Ergebnisse der Implementierung zeigen ferner, dass Abstraktionen behilflich sind. Würden die Prozeduren gemäß den tatsächlich benötigten Daten und mit Betrachtung der ISO 29002-31 angepasst, ließen sich zusätzliche Datenbankabfragen vermeiden und die Komplexität in der Business Logik Schicht verringern. Dennoch vereinfachen die Prozeduren den Zugriff enorm, denn wie Listing 5.13 zeigt, kapseln die Prozeduren komplexe Anfragen und vereinfachen die Schnittstelle zur Datenbank.

Das Fehlerhandling wird in den Schema-Modellen nicht betrachtet. Um Informationen beim Auftreten von Fehlern an den Klienten zu senden, wurde eine Fehler-Schemadatei

5. Implementierung

mit Fehlertypen erstellt und das Katalog-Schema angepasst, sodass als Antwort ggf. Fehlertexte mitgesendet werden können.

Auf die vorhandene Implementierung der Business Logik und der Modelle der Schemata lässt sich sehr einfach ein weiterer Webservice basierend auf SOAP implementieren. Das bedeutet, dass die oberste Schicht ausgetauscht werden kann. Es wurden beide Technologien verwendet, welche die gleiche Business Logik aufrufen.

Schlussfolgerung und Ausblick in die Zukunft

Die Zielsetzung dieser Arbeit ist, eine Lösung für die in der Regel unflexiblen und starren Produktdatenaustauschschnittstellen in der Industrie zu erarbeiten. Hierbei wurde erfolgreich als Prototyp eine flexible Abfrage- und Antwortschnittstelle analog zu einer SQL-Abfrage- und Antwortschnittstelle mit Hilfe der in der ISO 29002-31 beschriebenen Ontologie erstellt. Ferner wurde diese Schnittstelle an die PLIB als Datenbasis angebunden, sowie die bereitgestellten Prozeduren zum Datenaustausch genutzt. Es konnte festgestellt werden, dass mit aktuellen Techniken und Frameworks eine relativ schnelle Integration der den Standards zugehörigen Schemata möglich ist. Es lässt sich ein Webservice erstellen, welcher die Flexibilität einer Abfrage- und Antwortschnittstelle besitzt und auf Schemata basiert. Dabei wurde ein RESTful-Webservice erstellt, welcher die Abfrage entgegennimmt, für die Datenbankprozeduren transformiert und die Antwort entsprechend auf das Modell der Antwortnachricht transformiert. Sind diese Transformationslogiken implementiert, lassen sich darauf aufbauend weitere Schnittstellen implementieren wie in diesem Falle ein weiterer Webservice mittels SOAP-Protokoll.

Während der Analyse und Implementierung ergaben sich einige Problemstellungen, die gelöst werden konnten. Beispielsweise ist die Heterogenität der Datenstrukturen ein nicht triviales Problem. Die Prozeduren konnten nicht direkt mit den von der Abfrage erhaltenen Daten aufgerufen werden. Es sind vorab Anfragen zur Transformation an die Datenbank nötig. Die zurückgelieferten Daten der Prozeduren liefern nicht alle Daten zurück, die für das Füllen der XML-Katalog-Daten nötig sind. Ferner passt die Struktur und Semantik der zurückgelieferten Daten nicht mit der Antwort des Standards überein. Nicht zu erkennen sind auch technische Probleme, denn wird von der darüberliegenden Schicht ein Standard nicht unterstützt, so gilt es auch dafür eine Lösung zu finden. In diesem Fall wurden RECORD-Typen nicht von Java-JDBC unterstützt. Dies konnte durch entsprechende Absprache unter den Studenten des Fachbereiches gelöst werden, sodass an den Prozeduren und Rückgabetypen Änderungen durchgeführt wurden, welche das technische Problem lösen konnten.

Die Umsetzung der Arbeit erfolgte mit Hilfe aktueller Technologien auf Basis von REST Webservices. Die entwickelte Applikation kann automatisiert mit Hilfe eines Build Management Systems erzeugt werden. Der Vorteil ist das einfache und standardisierte Erzeugen der Applikation. Die Quelltexte sind in einem Versionskontrollsystem abgelegt, dies ermöglicht die genauen Entwicklungsschritte zu verfolgen und gibt die Möglichkeit auf ältere Versionsstände zu blicken oder diese zu erzeugen.

Die Arbeit umfasst nicht die Implementierung einer GUI, es wurde dennoch zu Testzwecken und zur Simulation eine einfache Benutzeroberfläche geschaffen, über die

5. Implementierung

XML-Abfragen abgesendet werden können und die Antwort angezeigt wird. Ebenfalls nicht Bestandteil der Arbeit ist die Implementierung einer Schnittstelle nach ISO 22745-30 Identification Guide. Dieser Standard beschreibt eine Möglichkeit, Regeln zu definieren, welche Konzepte (z.B aus PLIB) näher beschreiben. Das ermöglicht sinnvollerweise eine Einschränkung der Anfrageparameter in der Form, wie es der Kunde respektive Client in seinem Kontext benötigt. Man stelle sich vor, dass in einer Produktionsabteilung andere Daten von bestimmten Konzepten nötig sind als in der Verkaufsabteilung. Ein Beispiel wäre die Information über bestimmte Eigenschaften des Materials über die Verformung bei großer Hitze in der Produktion. Bei einem Metall ist diese Information in der Produktion wichtig, da hier ggf. hohe Hitzeentwicklung vorzufinden ist. Der Verkauf benötigt diese Informationen nicht oder nur gewisse Teile davon. Dies kann mit Hilfe von Identification Guides eingeschränkt werden. Somit wäre als Folgearbeit beispielsweise denkbar, die Implementierung von Identification Guides vorzunehmen und mit der in dieser Arbeit entwickelten Lösung zu verbinden. Um diese Einschränkung nach Regeln gemäß ISO 22745-30 zu demonstrieren, könnte eine Benutzeroberfläche aus den Angaben gemäß ISO 22745-30 generiert werden. Nehmen wir aus obigem Beispiel eine Applikation, welche eine Benutzeroberfläche für die Abteilung Produktion und eine, die die Abteilung Verkauf repräsentiert. Diese könnten nach ISO 22745-30 unterschiedlich definierte Regelwerke besitzen und folglich eine andere Maske darstellen. Die Abfrage, die somit von der Abteilung Produktion über die Maske gesendet werden kann, ermöglicht gewisse Eigenschaften eines Produktes abzufragen, die für die Abteilung relevant sind. Die Abfrage der Abteilung Verkauf kann folglich andere gleichsam zusätzliche oder ggf. weniger Eigenschaften abfragen, die nur für diese Abteilung relevant sind. Diese Abfragen werden dann eine Abfrage gemäß dieser Arbeit erzeugen, eingeschränkt gemäß des dynamisch erzeugtem Formular nach Identification Guide.

Als mögliche weitere Folgearbeit wäre eine Gesamtintegration aller Arbeiten im Fachbereich denkbar. Diese Arbeit könnte die Arbeiten von Herrn Mende, Herrn Loth, Frau Janßen und Herrn Sobek zu einer der Praxis entsprechenden Gesamtlösung integrieren. Hierdurch würde man Erkenntnisse über die Flexibilität der eingesetzten Standards erlangen.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Ort, Datum

Unterschrift (Stefan Sobek)

Literaturverzeichnis

- [Ben11] BENSON, Peter R.: *ISO 8000 Data Quality*. 2011
- [Coc00] COCKBURN, Alistair: *Writing Effective Use Cases*. 3. Addison-Wesley, 2000. – ISBN 978-0201702255
- [Dai12] DAIGNEAU, Robert: *Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services*. 1. Addison-Wesley, 2012
- [Fie00] FIELDING, Roy T.: *Architectural Styles and the Design of Network-based Software Architectures*. <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>, University of California, Irvine, Diplomarbeit, 2000
- [Hem09] HEMMJE, Matthias L.: *Informations- und Wissensmanagement im Internet*. Kurs 01853 - Fernuniversitaet in Hagen, 2009
- [ISO09a] ISO13584-42: *ISO 13584-42 - Industrial Automation Systems and Integration - Parts Library - Part 42: Methodology for structuring part families*. 2009
- [ISO09b] ISO22745-30: *ISO 22745-30 - Industrial automation systems and integration — Open technical dictionaries and their application to master data — Identification guide representation*. 2009
- [ISO09c] ISO29002-31: *ISO 29002-31 - Industrial automation systems and integration — Exchange of characteristic data - Query for characteristic data*. 2009
- [Jan13] JANSSEN, Sandra: *Implementierung von Web Services gemäß ISO 29002-20 zur Auflösung von Konzept- Identifikatoren in Konzept-Dictionaries /Ontologien*, Fernuniversität Hagen, Diplomarbeit, 2013
- [Six09] SIX, Hans-Werner: *Software Engineering I Methodische Entwicklung objektorientierter Desktop-Applikationen*. Kurs der Fernuni Hagen SS 2009, 2009
- [Spi11] SPILLER, Martin: *Maven* 3. Mitp-Verlag, 2011 <http://www.amazon.de/Maven-Konfigurationsmanagement-Java-mitp-Professional/dp/3826691180>. – ISBN 9783826691188
- [Sta09] STARKE, Gernot: *Effektive Software- Architekturen*. Hanser, 2009. – ISBN 978-3-446-42008-3
- [Swi08] SWICEGOOD, Travis: *Pragmatic Version Control using Git*. Pragmatic Bookshelf, 2008
- [Til09] TILKOV, Stefan: *REST und HTTP: Einsatz der Architektur des Web für Integrationszenarien*. Bd. 2. dpunkt.verlag, 2009
- [Uit12] UITERWYK, Henning: *Die Bedeutung der Merkmalleisten bei eCl@ss*. PROLIST open 2012, Karlsruhe, 2012

- [Wri02] WRIGHT, Brian: *Oracle9i SQLJ Developer's Guide and Reference - Release 2 (9.2)*. Website. http://docs.oracle.com/cd/B10501_01/java.920/a96655/toc.htm. Version: 2002

Anhang A.

Analyse ISO 29002-31 - Exchange of characteristic data

A.1. XML Datencontaineranalyse ISO 29002-31

Die Unterkapitel beschreiben die einzelnen XML-Datencontainer aus der ISO 29002-31. Der Ausgangspunkt ist der query_context, welcher einige Metadaten zum eigentlichen query enthält.

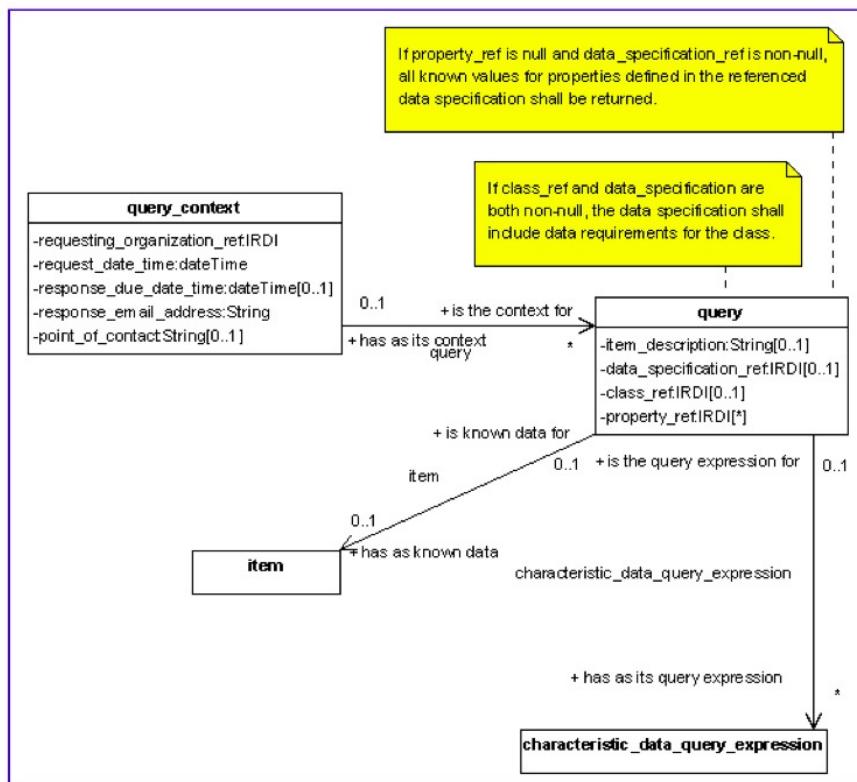


Abbildung A.1.: UML-Diagramm Query Main¹

A.1.1. query_context

Dies ist eine Art Container für eine Menge von Queries. Inhalt sind Informationen über den Anforderer der Daten zwecks persönlicher Kontaktaufnahme, wie z.B. die Anfrage-

¹Quelle: ISO 29002-31 Kapitel 5.2.1

zeit, Informationen über die Organisation, welche die Anfrage schickt, sowie einen gewünschten Antwortzeitpunkt mit Antwort-E-mail Adresse. Siehe dazu Abbildung A.1 und (vgl ISO09c, Kap. 5.2.2).

Da die Vorgabe lautet, den Service auf Basis eines Webservices zu erstellen, entfällt die Benutzung des query_context. Der Grund ist, dass der Kontext implizit durch den Webservice respektive dem Server zur Verfügung gestellt wird. Beispielsweise wird die Anfragezeit zwar nicht explizit durch den Serviceaufrufer selbst übergeben, allerdings durch die Anfrage an den technischen Server wie z.B. Apache Tomcat Server mittels Logeintrag implizit ermittelt. Somit lassen sich diese Metadaten über Verbindungsprotokolle der Infrastruktur herausfinden. Siehe dazu auch (Kap. 6 ISO09c), welche besagt: »ISO/TS 29002 can be implemented:

- a. with another envelope standard, such as EDI, or
- b. by itself, using the query_context to carry envelope information.«

A.1.2. query

Die Unterstützung aller Funktionalitäten des queries entspricht laut ISO 29002-31 der Conformance class 1: simple query (Anhang 6 ISO09c). Dies ist der eigentliche Abfrage-Datensatz. Abgefragt werden kann mittels class IRDI², data_specification IRDI, eine Menge von property IRDI, Teiledaten (das sind Teile gefüllt mit Daten ihrer Eigenschaften, die dem Klienten bereits bekannt sind) und einer item_description. Das bedeutet, dass bereits bekannte Eigenschaften eines Teils übertragen werden können, um die Suche auf Teile mit diesen Werte-Eigenschaften einzuschränken.

Die data_specification IRDI verweist auf eine Spezifikation aus ISO 22745-30, die besagt welche Properties für dieses Teil sinnvoll sind. Die angegebenen Property IRDIs sind dann eine Teilmenge aus den mittels data_specification IRDI definierten erlaubten Eigenschaften. Für weitere Informationen zur ISO 22745-30 siehe Abschnitt A.2.

Denkbar sind einfache Abfragen wie z.B.: »Gib mir alle Teile der Klasse xyz«. Mitgeliefert werden auch Teile von Subklassen. Weiterhin kann die Abfrage nach bestimmten Eigenschaften eingeschränkt werden. Eine weitere Möglichkeit ist es, bereits bekannte Daten über ein Teil zu übermitteln, mit dem Zwecke hierüber die IRDI, zu erfahren oder weitere Eigenschaftsdaten zu erhalten. Siehe Beispielqueries simple queries in Unterabschnitt A.1.4.

A.1.3. characteristic_data_query_expression (parametric_query)

Das entspricht laut ISO 29002-31 Anhang 6 der Conformance class 2: parametric query.

²IRDI - International registration data identifier

³Quelle: ISO 29002-31 Kapitel 5.3.1

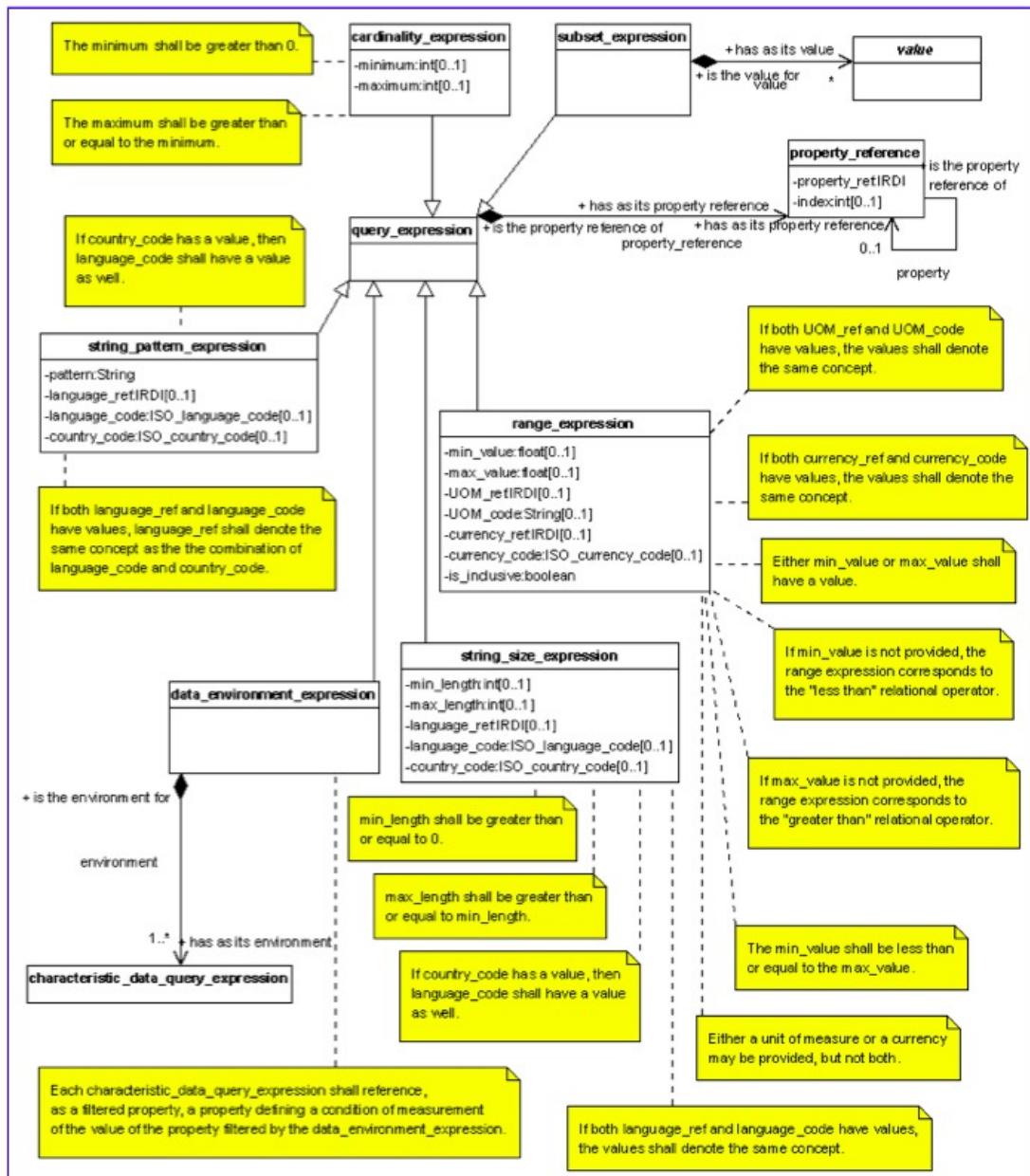


Abbildung A.2.: UML-Diagramm Query Expression³

Eine characteristic_data_query_expression kann verschiedene expressions vom Typ query_expression beinhalten. Von jedem Typ jeweils nur maximal eine. Z.B.

- string_size_expression
- string_pattern_expression
- range_expression
- data_environment_expression
- cardinality_expression
- subset_expression

Darüberhinaus noch folgende Attribute:

- property_reference - die property auf den die query_expression bezogen ist

Solch eine Expression ermöglicht das Filtern, gleichsam ein Einschränken bestimmter Properties und Werte.

A.1.4. Query Beispiele

Nachfolgend seien einige Query-Beispielszenarien aufgestellt, die sich aus der Analyse der Standards ergeben.

Eine Schraube hat die folgenden möglichen Eigenschaften:

Klassen-Identifier 1234-abcd# ab-cdefgh# 1 (IRDI)

Typ M6 (Property IRI: 1234-abcd# ab-bbbbbbb# 1)

Länge 80mm (Property IRI: 1234-abcd# ab-cccccc# 1)

Simple Query

Ein simpler query ermöglicht folgende Abfrage: »Gib mir alle Teile zum Konzept Kreuzschraube mit dem Identifier (IRDI) 1234-abcd#ab-cdefgh#1«. Das Ergebnis ist ein Teil, mit allen Attributen wie oben angegeben.

Ein anderer Query könnte lauten: »Gib mir die Properties 1234-abcd#ab-cccccc#1 und 1234-abcd#bbbbbb# 1 des Teils der Klasse 1234-abcd#ab-cdefgh#1«. Das Ergebnis wäre das Teil mit Typ: M6 und der Länge: 80mm.

Es könnte auch mit Hilfe von vorhandenen Daten gesucht werden, z.B.: »Hier ist ein Teil mit der Property Typ: M6 (Property IRI: 1234-abcd# ab-bbbbbbb# 1), gib mir bitte dazu die Properties 1234-abcd#ab-cccccc#1 und 1234-abcd#bbbbbb#1«

Parametric Query

Hat man jetzt noch eine Schraube mit folgenden Eigenschaften:

Klassen-Identifier 1234-abcd#ab-cdefgh#1 (IRDI)

Typ M5 (Property IRDI: 1234-abcd#xx-bbbbbbb#1)

Länge 100mm (Property IRDI: 1234-abcd#xx-cccccc#1)

ermöglicht der Parametric Query mit Hilfe der characteristic_data_query_expression folgende Abfragen: »Gib mir die Properties 1234-abcd# ab-cccccc#1 (Länge) und 1234-abcd#bbbbbb#1 (Typ) des Konzeptes 1234-abcd#ab-cdefgh#1 (Schraube) mit einer Länge zwischen 50 und 150mm und dem Typen M5 oder M6.«

Dies ermöglicht das Filtern auf genau eine übergebene Property. Rekursive Abfragen sind auch möglich, beispielsweise wenn die gesuchte Property eine Multi-Property ist (Property: Loch als Wert zwei Properties mit Form und Durchmesser und Durchmesser soll gefiltert werden).

A.2. Analyse ISO 22745-30 - Identification Guide

Ein Identification Guide beschreibt, welche Daten für ein Objekt benötigt werden, damit dies überhaupt sinnvoll für einen bestimmten Zweck eingesetzt werden kann. Der Käufer, Produktmanager oder Benutzer definiert die Anforderungen an die Daten. Ein »Datenanforderungsstatement« wird als ein i-xml Identification Guide xml file erzeugt (vgl. Ben11, Slide 14 - Automating the Data Supply Chain). Es wird die Frage beantwortet, welche Daten (Properties) zu einem bestimmten Konzept eines Objektes benötigt werden, um den Artikel zu kaufen oder zu sinnvoll zu verwalten. Diese Anforderungen werden von der Abfrageseite (Kundenseite) definiert, also derjenige, der Daten abfragen möchte⁴. Ein Identification Guide referenziert Konzepte eines Dictionaries, um Datenanforderungen einer bestimmten Klasse zu beschreiben (vgl. ISO09b, Kapitel 5). Ein Datenempfänger kann eine Organisation oder eine Gruppe von Organisationen oder Firmen sein, welche ähnliche Datenanforderungen haben. Somit wird eine Identification Guide Gruppe von einer speziellen Organisation verwaltet, welche wiederum selbst Datenempfänger sein kann.

⁴Quelle: ECCMA_ISO_8000_certification.pdf - Zertifizierungspräsentation der ECCMA zur ISO 8000

Anhang B.

Anwendungsfälle

Dieses Kapitel beinhaltet weitere Anwendungsfälle des Systems. Der in Abschnitt 3.3 bereits beschriebene Anwendungsfall wird hier nicht erneut aufgeführt.

B.0.1. Charakteristische Daten eines Produkts validieren

use case Charakteristische Daten validieren

actors

Klient

precondition

Der Klient verwendet einen gültigen Identifier sowie auf den Identifier passende Daten.

main flow

Der Klient gibt einen Identifier eines Teils ein. Zusätzlich übermittelt er zu diesem bekannten Teil Eigenschaften dieser Instanz des Teils und sendet eine Anfrage ab. Die Anfrage wird auf Gültigkeit überprüft. Als Antwort bekommt er ein oder mehrere Datensätze von Teilen mit den entsprechenden charakteristischen Daten zurück, auf welche die übergebenen Eigenschaften zutreffen.

postcondition

Alle Daten aller Teile der gewählten Klassen des Identifiers werden zurückgegeben. Dies ermöglicht dem Klienten eine Validierung der ihm bereits bekannten Daten über ein Element.

alternative flow

Die übermittelten Eigenschaften des Teils stimmen nicht mit den gespeicherten Daten überein.

postcondition

Es werden keine Daten zurückgeliefert. Die übermittelten Daten sind nicht valide.

exceptional flow Ungültige Identifier oder ungültige Anfrage

Der oder die übergebenen Identifier oder die gesamte Anfrage ist gemäß Spezifikation ungültig.

postcondition

Es wird eine Fehlermeldung zurückgegeben.

end Charakteristische Daten validieren

Beispiel

In diesem Anwendungsfall verfügen wir bereits über Teile/Wertepaare eines bestimmten Konzeptes, z.B. eben jenen Schraubendreher.

»Ich habe hier ein mir bekanntes Teils mit bestimmten Eigenschaften (Properties), Länge=300mm. Gib mir alle Teile und alle Properties der Klasse mit dem Identifier 0173-1#01-AAA352#4 (Kreuzschraube), welche die mitgelieferten Eigenschaften haben.« Das Ergebnis sind Teile mit allen Properties des angegebenen Konzeptes, welche über die übergebenen Eigenschaften (Properties) verfügen. In unserem Fall vervollständigen wir unsere Properties mit den weiteren Properties »Typ« und »Spannungsfest«.

Die XML-Abfrage gemäß query.xsd¹ sieht so aus:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <qry:query xsi:schemaLocation="...query query.xsd" xmlns:xsi="http://www.w3.org
   /2001/XMLSchema-instance" xmlns:cat="...catalogue" xmlns:val="...value"
   xmlns:qry="...query" xmlns:bas="...basic">
3   <cat:item class_ref="0173-1#01-AAA352#4..">
4     <cat:property_value property_ref="0173-1#01-BBB111#1">
5       <val:integer_value></val:integer_value>
6     </cat:property_value>
7   </cat:item>
8 </qry:query>
```

Listing B.1: Query Beispiel - Daten validieren

B.0.2. Charakteristische Daten mittels Suchausdruck abfragen

use case Charakteristische Daten mit Suchausdruck abfragen

actors

Klient

precondition

Der Klient verwendet einen gültigen Identifier.

main flow

Der Klient gibt einen Identifier eines Konzeptes ein. Ferner übergibt er ein oder mehrere bekannte Property Identifier sowie passend dazu Werte zur Sucheinschränkung.

postcondition

Alle Elemente auf jene diese Einschränkung der übergebenen Werte zutrifft werden zurückgegeben.

alternative flow Keine Werte zur Werteeinschränkung gefunden

Die übermittelten Werte der mittels Property Identifier identifizierten Eigenschaftswerte sind nicht zum Teile gespeichert.

postcondition

Es werden keine Daten zurückgeliefert.

¹Schema Datei ist referenziert in ISO 29002-31, liegt der Arbeit bei

exceptional flow Ungültige Identifier oder ungültige Anfrage

Der oder die übergebenen Identifier oder die gesamte Anfrage ist gemäß Spezifikation ungültig.

postcondition

Es wird eine Fehlermeldung zurückgegeben.

end Charakteristische Daten mit Suchausdruck abfragen

Beispiel

Wir nehmen das Schraubendreher Beispiel aus Abschnitt 3.3.3 zur Hand, und möchten eine Abfrage absenden, welche von der Klasse Schraubendreher alle Items erhalten soll die eine Länge zwischen 200 und 300 mm haben.

Um nun alle Eigenschaften (Properties), wie Länge, Typ und Spannungsfest zu erhalten muss folgende Abfrage gesendet werden: »Gib mir alle Items und alle Properties der Klasse mit dem Identifier 0173-1#01-AAA352#4 (Kreuzschraube).« Das Ergebnis ist ein Item mit allen Attributen (Properties) der gewünschten Klassen und gegebenenfalls vorhandenen Unterklassen. In unserem Falle genau die oben angegebenen Werte.

Die XML-Abfrage gemäß query.xsd² sieht so aus:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <query xsi:schemaLocation="...query query.xsd" xmlns:xsi="http://www.w3.org
   /2001/XMLSchema-instance" xmlns:cat="...catalogue" xmlns:val="...value"
   xmlns:qy="...query" xmlns:bas="...basic">
3    <class_ref>0173-1#01-AAA352#4</class_ref>
4    <characteristic_data_query_expression>
5      <range>
6        <property_reference property_ref="0173-1#01-BBB111#1"/>
7        <min_value>200</min_value>
8        <max_value>300</max_value>
9        <is_inclusive>true</is_inclusive>
10      </range>
11    </characteristic_data_query_expression>
12  </query>
```

Listing B.2: Query Beispiel - Daten mit Suchausdruck abfragen

²Schema Datei ist referenziert in ISO 29002-31, liegt der Arbeit bei

Anhang C.

Installationsanleitung Apache Tomcat 7

C.1. Installation auf Mac OS 10.8

Diese Installationsanleitung bezieht sich auf die Installation des Apache Tomcat 7 in Mac OS 10.8.

C.1.1. Prüfen der Java Version

Mittels

```
java -version
```

prüfen, ob Version 1.6 oder 1.7 installiert ist. Falls nicht, muss Java vorher installiert werden. Dazu das Java Development Kit herunterladen und installieren:
<http://www.oracle.com/technetwork/java/javase/overview/index.html>

Für den Mac ist das ggf. über die Apple-Webseite verfügbar.

C.1.2. Tomcat herunterladen und entpacken

- Auf <https://tomcat.apache.org/download-70.cgi> Tomcat herunterladen. Darauf achten eine Binary distribution herunterzuladen, z.B. apache-tomcat-7.0.42.tar.gz
- Das Paket entpacken mittels tar -xvzf apache-tomcat-7.0.42.tar.gz
- Ein Verzeichnis unter /usr/local erstellen, wo der Apache später laufen soll, danach die Dateien dort hinkopieren
 - sudo mkdir -p /usr/local
 - sudo mv /Downloads/apache-tomcat-7.0.42 /usr/local/
- Einen symbolischen Link erstellen um später einfacherer zwischen Versionen umzuschalten:
 - sudo rm -f /Library/Tomcat
 - sudo ln -s /usr/local/apache-tomcat-7.0.42 /Library/Tomcat
- Allen Apache Dateien den aktuellen User als Besitzer setzen und Rechte vergeben:

- sudo chown -R <dein_username> /Library/Tomcat
- sudo chmod +x /Library/Tomcat/bin/*.sh

C.1.3. Tomcat starten

/Library/Tomcat/bin/startup.sh

C.1.4. Tomcat stoppen

/Library/Tomcat/bin/shutdown.sh

C.2. Installation unter Windows

Am besten wird der Tomcat mittels Installer installiert. Hier wird man durch eine grafische Benutzeroberfläche geführt und Apache Tomcat wird als Dienst in das System integriert.

C.3. Tomcat mittels Maven starten

Um Tomcat mittels Maven zu starten, muss Tomcat als Plugin in der Maven pom.xml konfiguriert werden. Das Listing C.1 zeigt die Konfiguration für die pom.xml.

```
1 <build>
2   <plugins>
3     <plugin>
4       <groupId>org.apache.tomcat.maven</groupId>
5       <artifactId>tomcat7-maven-plugin</artifactId>
6     </plugin>
7   </plugins>
8 </build>
```

Listing C.1: Tomcat 7 Maven Plugin

Anschließend kann der Tomcat-Server mit folgendem Befehl gestartet werden:

```
mvn tomcat7:run
```

Anhang D.

Automatische Entwicklertests

D.1. Unit Test

Nachfolgend ein Beispiel eines Unit Tests. Das Testobjekt ist die Klasse XMLMarshaler, welcher für das Marshalling und Unmarshalling der XML-Daten verantwortlich ist. Hinweis: Alle Quelltexte verzichten auf import- und package-Anweisungen um die Lesbarkeit zu erhöhen. Die Original Quelltexte liegen der Arbeit bei.

```
1 package de.feuplib.xml;
2
3 /**
4  * Tests the marshalling and unmarshalling of xml files.
5 */
6 public class XMLMarshallerTest extends AbstractXMLTest {
7
8     /** XML Marshaller instance under test */
9     XMLMarshaller marshaller;
10
11    /** Logger instance */
12    private static final Logger LOGGER = Logger.getLogger(XMLMarshallerTest.class)
13        ;
14
15    /**
16     * Simple marshalling test with arbitrary catalogue item.
17     */
18    @Test
19    public void testMarshallingWithValidArbitraryCatalogue() {
20        String catalogue = marshaller.marshall(createCatalogueWithOneItem());
21        LOGGER.info(catalogue);
22        assertTrue(catalogue.contains("0173-1#01-AAA352#4"));
23        assertTrue(catalogue.contains("true"));
24    }
25
26    /**
27     * Simple test with unmarshalling an arbitrary item from xml.
28     */
29    @Test
30    public void testUnMarshallingWithValidArbitraryClassIrdi() {
31        QueryType queryType = marshaller.unmarshallXML(readXMLFrom("/de/feuplib/
32            xml/query_class_irdi.xml"),
33            QueryType.class);
34        assertEquals("0173-1#01-BAD803#2", queryType.getClassRef());
35    }
36
37    /**
38     * Throws an exception while parsing on illegal irdi passed.
39     */
40     * @throws Exception on illegal irdi
41     */
```

```

40     @Test(expected = RuntimeException.class)
41     public void shouldThrowExceptionDuringXMLValidationWithIllegalIrdi() throws
42         Exception {
43         QueryType queryType = marsteller.unmarshallXML(readXMLFrom("/de/feu/plib/
44             xml/query_class_irdi_illegal.xml"),
45             QueryType.class);
46     }
47
48     /**
49      * creates a sample catalogue for testing
50      *
51      * @return the sample catalogue with one item
52      */
53     private CatalogueType createCatalogueWithOneItem() {
54         ItemType item = new ItemType();
55         item.setClassRef("0173-1#01-AAA352#4");
56         PropertyValue propertyValueType = new PropertyValue();
57
58         BooleanValueType bvt = new BooleanValueType();
59         bvt.setValue(true);
60         propertyValueType.setBooleanValue(bvt);
61         propertyValueType.setPropertyRef("0173-1#01-A35AA2#4");
62         item.getPropertyValue().add(propertyValueType);
63         CatalogueType catalogue = new CatalogueType();
64         catalogue.getItem().add(item);
65
66         return catalogue;
67     }
68
69     /**
70      * @throws java.lang.Exception
71      */
72     @Before
73     public void setUp() {
74         marsteller = new XMLMarshallerImpl();
75     }
76
77     /**
78      * @throws java.lang.Exception
79      */
80     @After
81     public void tearDown() {
82         marsteller = null;
83     }

```

Listing D.1: Beispiel eines Unit Tests

Das Listing D.2 zeigt die abstrakte Klasse »AbstractXMLTest«, welche Hilfsmethoden für XML-Tests kapselt. Hier kapselt diese Klasse die Funktionalität XML-Dateien einzulesen.

```

1 /**
2  * Abstract test class. Extend from this class to get functionality to read XML
3  * files for your test.
4 */
5 public class AbstractXMLTest {
6

```

```

7   protected XMLMarshaller marshaller;
8
9   /**
10    * Logger instance
11    */
12   private static Logger LOGGER = Logger.getLogger(AbstractXMLTest.class);
13
14   @Before
15   public void setUp() {
16       marshaller = new XMLMarshallerImpl();
17   }
18
19   @After
20   public void tearDown() {
21       marshaller = null;
22   }
23
24   /**
25    * Reads the xml file from given filename
26    *
27    * @param filename the filename of the xml
28    * @return the string content of the xml file
29    */
30   protected String readXMLFrom(String filename) {
31       BufferedReader br = null;
32       StringBuffer sb = new StringBuffer();
33
34       try {
35           String currentLine;
36
37           InputStream is = XMLMarshallerImpl.class.getResourceAsStream(filename)
38           ;
39           br = new BufferedReader(new InputStreamReader(is));
40
41           while ((currentLine = br.readLine()) != null) {
42               sb.append(currentLine);
43           }
44
45           } catch (IOException e) {
46               e.printStackTrace();
47           } finally {
48               try {
49                   if (br != null) br.close();
50               } catch (IOException ex) {
51                   LOGGER.info("Exception occurred during reading file: " + ex);
52               }
53           }
54
55           return sb.toString();
56       }
57   }

```

Listing D.2: Abstrakte Unit Testklasse

D.2. Integrationstest

Dieser Abschnitt zeigt beispielhaft einen Integrationstest auf. Der Integrationstest unterscheidet sich vom Unit-Test darin, dass eine Komponente integriert mit mehreren abhängigen Komponenten getestet wird. Der nachfolgende Integrationstestfall »PlibDaoIT« aus Listing D.3 testet die Klasse »PlibDao«. Es ist ein Integrationstest, da für den Test die gesamte Datenbank benötigt wird. Nur so kann sinnvoll das Datenzugriffssobjekt getestet werden.

Gestartet werden kann der Integrationstest mit Maven mittels Aufruf von

```
mvn integration-test
```

```
1  /**
2  * Integration Test of PLIB Dao
3  */
4
5 @RunWith(SpringJUnit4ClassRunner.class)
6 @ContextConfiguration(locations = {"beans_for_tests.xml"})
7 public class PlibDaoIT {
8
9     /**
10      * Logger instance
11      */
12     private static Logger LOGGER = Logger.getLogger(PlibDaoIT.class);
13
14     @Autowired
15     private PlibDao plib;
16
17     @Test
18     public void shouldReturnTrueWithExistingIRDI() {
19         Irdi irdi = createMultiListTestIrdi();
20         assertTrue(plib.doObjectsExistsWithThis(irdi));
21     }
22
23     @Test
24     public void shouldReturnFalseWithIRDINotinDB() {
25         Irdi irdi = createNonExistingTestIrdi();
26         assertFalse(plib.doObjectsExistsWithThis(irdi));
27     }
28
29     @Test
30     public void shouldReturnFalseWithEmptyIRDI() {
31         Irdi irdi = new Irdi() {
32             @Override
33             public String getIrdi() {
34                 return "";
35             }
36         };
37         assertFalse(plib.doObjectsExistsWithThis(irdi));
38     }
39
40     @Test
41     public void testGetNumberOfObjectsOfIrdiExistingIrdi() {
42         Irdi irdi = createMultiListTestIrdi();
43         assertEquals(8, plib.getNumberOfObjectsOfIrdi(irdi));
44     }
45 }
```

```

46     @Test
47     public void testGetNumberOfObjectsOfIrdiNotExistingIrdi() {
48         Irdi irdi = createNonExistingTestIrdi();
49         assertEquals(0, plib.getNumberOfObjectsOfIrdi(irdi));
50     }
51
52     @Test
53     public void testThatWhenReadExternalProductIdsByIrdiMustReturnSome() throws
54         Exception {
55         Irdi irdi = createMultiListTestIrdi();
56         List<BigDecimal> productIds = plib.readExternalProductIdsBy(irdi);
57         assertNotNull(productIds);
58         assertEquals(8, productIds.size());
59     }
60
61     /**
62      * Currently there are two instances in the database, seems that these are
63      * duplicates but not sure.
64      */
65     @Test
66     public void shouldReturnOneTestExternalIdWithTestIrdi() {
67         Irdi irdi = createSkalpellIrdi();
68         List<BigDecimal> externalProductIds = plib.readExternalProductIdsBy(irdi);
69         assertNotNull(externalProductIds);
70         assertEquals(2, externalProductIds.size());
71         LOGGER.info("external product ids: " + externalProductIds.get(0));
72         assertEquals(new BigDecimal("300000001"), externalProductIds.get(0));
73     }
74
75     /**
76      * This is a bigger integration test.
77      * <ul>
78      *   <li>First create an irdi instance</li>
79      *   <li>create an enriched query</li>
80      *   <li>Then load the objects from the database with its properties</li>
81      *   <li>There should be the same number ob objects than with the previous
82      *       check</li>
83      * </ul>
84      */
85     @Test
86     @Ignore("loadObjectsFrom is obsolete, maybe later reimplemented")
87     public void shouldReturnOneInstanceOfSkalpellWithTwoProperties() {
88         Irdi skalpellIrdi = createSkalpellIrdi();
89
90         List<BigDecimal> externalProductIds = plib.readExternalProductIdsBy(
91             skalpellIrdi);
92
93         EnrichedQuery query = createEnrichedQueryFrom(skalpellIrdi);
94         query.setType(QueryKind.SIMPLE);
95         CatalogueType catalogueType = plib.loadObjectsFrom(query);
96         List<ItemType> itemTypes = catalogueType.getItem();
97         assertEquals(externalProductIds.size(), itemTypes.size());
98         assertEquals(1, itemTypes.get(0).getPropertyValue().size());
99     }
100
101    /**
102     * Test should return two items where the first one would be checked.
103     * Should have two properties.
104     */
105    @Test
106    public void testLoadPropertiesByExternalIds() {

```

```

103     List<BigDecimal> externalIds = new ArrayList<BigDecimal>();
104     BigDecimal bigDecimal = new BigDecimal("300000001");
105
106     externalIds.add(bigDecimal);
107     List<List<Map<String, Object>>> valueTypeList = plib.
108         loadStringPropertiesByExternalIds(externalIds);
109     LOGGER.info("valuetype list: " + valueTypeList);
110     assertEquals("should be one instance", 1, valueTypeList.size());
111     assertEquals("should be two properties", 2, valueTypeList.get(0).size());
112     assertThatIrdiAndValueAreAvailable(valueTypeList.get(0).get(0).entrySet(),
113         "0173-1#02-AAA762#1");
114     assertThatIrdiAndValueAreAvailable(valueTypeList.get(0).get(1).entrySet(),
115         "0173-1#02-AAB011#1");
116 }
117
118 private void assertThatIrdiAndValueAreAvailable(Set<Map.Entry<String, Object>>
119     entrySet, String knownIRDI) {
120
121     Set<Map.Entry<String, Object>> entries = entrySet;
122     boolean irdiFound = false;
123     boolean valueFound = false;
124     for (Map.Entry<String, Object> entry : entries) {
125         LOGGER.info("key of property: " + entry.getKey());
126         LOGGER.info("value of property: " + entry.getValue());
127         if ("IRDI".equals(entry.getKey()) && null != entry.getValue() && !"null".equals(entry.getValue())) {
128             irdiFound = true;
129             assertEquals(knownIRDI, entry.getValue());
130         }
131     }
132     assertTrue(irdiFound && valueFound);
133 }
134
135 /**
136  * Load data type and unit for skalpell length property which should be mm and
137  * mm as well.
138  * Property id: 300903090000033914 and 300903090000034450
139  */
140 @Test
141 public void testLoadDataTypeAndUnitForAPropertyById() {
142     List<Map<String, Object>> propertyTypeAndUnit = plib.
143         loadTypeAndUnitOfPropertyBy("300903090000033914");
144     LOGGER.info("property size: " + propertyTypeAndUnit.size());
145     assertTrue(propertyTypeAndUnit.size() == 1);
146
147     assertThatUnitAndSubTypeAreAvailable(propertyTypeAndUnit);
148 }
149
150 private void assertThatUnitAndSubTypeAreAvailable(List<Map<String, Object>>
151     propertyTypeAndUnit) {
152     Set<Map.Entry<String, Object>> entries = propertyTypeAndUnit.get(0).
153         entrySet();
154
155     /*

```

```

153     * we need to assure that we have the unit (in this case SYMBOL, e.g. mm or
154     * m or cm)
155     * in the values as well as the type of the value (in this case
156     * real_measure_type)
157     */
158     String property_unit_symbol = "SYMBOL";
159     String property_type = "SUB_TYPE";
160     boolean unitfound = false;
161     boolean typefound = false;
162     for (Map.Entry<String, Object> entry : entries) {
163         LOGGER.info("property key: " + entry.getKey());
164         LOGGER.info("property value: " + entry.getValue());
165         if ("SYMBOL".equals(entry.getKey()) && null != entry.getValue() && !"null".equals(entry.getValue())) {
166             unitfound = true;
167         }
168         if ("SUB_TYPE".equals(entry.getKey()) && null != entry.getValue() && !"null".equals(entry.getValue())) {
169             typefound = true;
170         }
171     }
172     assertTrue(unitfound && typefound);
173
174     private EnrichedQuery createEnrichedQueryFrom(Irdi skalpellIrdi) {
175         QueryType queryType = new QueryType();
176         queryType.setClassRef(skalpellIrdi.getIrdi());
177         return new EnrichedQuery(queryType);
178     }
179
180     private Irdi createNonExistingTestIrdi() {
181         return new Irdi() {
182             @Override
183             public String getIrdi() {
184                 return "0141-1#01-xxx#1";
185             }
186         };
187     }
188
189     private Irdi createMultiListTestIrdi() {
190         return new Irdi() {
191             @Override
192             public String getIrdi() {
193                 return "0141-1#01-UKU1#1";
194             }
195         };
196     }
197     /**
198      * Creates the irdi of an item which was created by me as testdata.
199      * It is a Skalpell (PREFERRED_NAME of class in DB)
200      * @return the irdi of the test item skalpell.
201     */
202     private Irdi createSkalpellIrdi() {
203         return new Irdi() {
204             @Override
205             public String getIrdi() {
206                 return "0173-1#01-BAD803#2";
207             }
208         };
209     }

```

Listing D.3: Beispiel eines Integrationstests

Anhang E.

Testszenarios und Testtools

Anbei die Testprotokolle der manuellen Entwicklertests.

Diese können mit den in Abschnitt E.3 aufgeführten Testtools oder alternativ mit der Testbenutzeroberfläche durchgeführt werden. Diese ist erreichbar unter:
<http://localhost:8080/plib-characteristic-query/query.xhtml>

E.1. Simple query

E.1.1. Vorhandene Teile anhand IRDI abfragen

Es wird eine IRDI übermittelt.

Vorbedingungen • Die Datenbankverbindung aufbauen: Oracle Datenbank muss gestartet werden.

Eingabedaten Testdatei simple_query_irdi.xml.

Durchführung • XML-Datei wird mittels curl Befehl an den Server gesendet.

• curl -v -H 'Content-Type: application/xml' -X POST -data '@simple_query_irdi.xml' http://localhost:8080/plib-characteristic-query/rest/ws/query

Erwartetes Ergebnis Zwei Items mit je zwei Eigenschaften werden erwartet.

Tatsächliches Ergebnis Zwei Items mit zwei Eigenschaften.

OK/Nicht OK? OK

E.1.2. Ungültige IRDI angegeben

Es wird eine ungültige IRDI (gemäß XSD ungültig) übermittelt.

Vorbedingungen • Die Datenbankverbindung aufbauen: Oracle Datenbank muss gestartet werden.

Eingabedaten Testdatei simple_query_illegal_irdi.xml.

Durchführung • XML-Datei wird mittels curl Befehl an den Server gesendet.

- curl -v -H 'Content-Type: application/xml' -X POST -data '@simple_query_illegal_irdi.xml'
http://localhost:8080/plib-characteristic-query/rest/ws/query

Erwartetes Ergebnis Fehlermeldung

Tatsächliches Ergebnis Fehlermeldung, marshalling error

OK/Nicht OK? OK

E.1.3. Vorhandene Teile anhand IRDI abfragen mit Projektion

Es wird eine IRDI übermittelt und zusätzlich eine Projektion (Eigenschaftsauswahl) vorgenommen.

Vorbedingungen • Die Datenbankverbindung aufbauen: Oracle Datenbank muss gestartet werden.

Eingabedaten Testdatei simple_query_projection_one_property.xml.

Durchführung • XML-Datei wird mittels curl Befehl an den Server gesendet.

- curl -v -H 'Content-Type: application/xml' -X POST -data '@simple_query_projection_one_property.xml'
http://localhost:8080/plib-characteristic-query/rest/ws/query

Erwartetes Ergebnis Zwei Items mit je einer Property, die angefragte Eigenschaft wird zurückgegeben.

Tatsächliches Ergebnis Zwei Items mit einer Property.

OK/Nicht OK? OK

E.1.4. Vorhandene Teile mit bekannten Werten validieren

Es wird eine IRDI übermittelt und zusätzlich Werte dieses Items übermittelt. Diese sollen validiert werden. Es wird dabei geprüft ob die Werte auch so in der Datenbank vorhanden.

Vorbedingungen • Die Datenbankverbindung aufbauen: Oracle Datenbank muss gestartet werden.

Eingabedaten Testdatei simple_query_validation.xml.

Durchführung • XML-Datei wird mittels curl Befehl an den Server gesendet.

- curl -v -H 'Content-Type: application/xml' -X POST -data '@simple_query_validation.xml'
http://localhost:8080/plib-characteristic-query/rest/ws/query

Erwartetes Ergebnis Die Werte des Items stimmen überein, daher wird exakt das Item zurückerwartet.

Tatsächliches Ergebnis Exakt das übergebene Item wird zurückgegeben, somit erfolgreich validiert.

OK/Nicht OK? OK

E.2. Parametric query

E.2.1. Abfrage mit Werteeinschränkung auf eine Eigenschaft eines Items

Es wird eine IRDI übermittelt und zusätzlich eine Sucheinschränkung auf einen Eigenschaftswert dieses Teils vorgenommen. In diesem Falle sollen TeilTeile gefunden werden, dessen Eigenschaftswerte sich im Bereich der übergebenen Werte befinden (Range).

Vorbedingungen • Die Datenbankverbindung aufzubauen: Oracle Datenbank muss gestartet werden.

Eingabedaten Testdatei parametric_query_range.xml.

Durchführung • XML-Datei wird mittels curl Befehl an den Server gesendet.

```
• curl -v -H 'Content-Type: application/xml' -X POST -data  
  '@parametric_query_range.xml'  
  http://localhost:8080/plib-characteristic-query/rest/ws/query
```

Erwartetes Ergebnis Es wird nur ein Wert von zwei der eingeschränkten Eigenschaft zurückgegeben.

Tatsächliches Ergebnis Nur ein Wert wurde zurückgegeben.

OK/Nicht OK? OK

E.3. Testtools

Zum manuellen Testen werden verschiedene Tools verwendet. Diese müssen in der Lage sein, einen POST-Request an eine URL zu senden und ein XML, das Query-XML, als Payload mitzusenden.

E.3.1. CURL

Auf Linux-basierten Systemen kann das Werkzeug »curl« verwendet werden. Dieses Werkzeug liegt allen Linux oder Unix basierten Betriebssystemen in der Regel bei, oder kann einfach nachinstalliert werden.

Je nach Betriebssystem wird das mittels Befehlszeile über einen sogenannten Paketmanager durchgeführt:

Ubuntu, Debian apt-get install curl

Fedora, RedHat, CentOS yum install curl

Ein Beispielaufruf ist in Listing E.1 zu sehen. Hier wird eine XML-Datei namens query class irdi.xml an die URL <http://localhost:8080/rest/ws/query/> gesendet.

```
1 curl -v -H "Content-Type: application/xml" -X POST --data "@src/test/resources/de/feu/plib/xml/query class irdi.xml" http://localhost:8080/rest/ws/query
```

Listing E.1: CURL Test des REST Webservices

E.3.2. Advanced REST Client

Eine andere Möglichkeit den REST Webservice zu testen ist ein Plugin des Browsers Chrome, genannt »Advanced REST Client«. Dieser kann auch einen POST-Request erzeugen und an eine URL senden. Die Abbildung E.1 zeigt einen Test eines einfachen XML-Files. Dazu muss der XML Inhalt in das untere Eingabefeld unter dem Reiter »Payload« eingeben werden. Ferner muss der Content-Type auf »application/xml« gesetzt werden, um entsprechende HTTP-Header mitzuschicken, welche dem Webservice mitteilen, dass auch tatsächlich XML-Daten kommen.

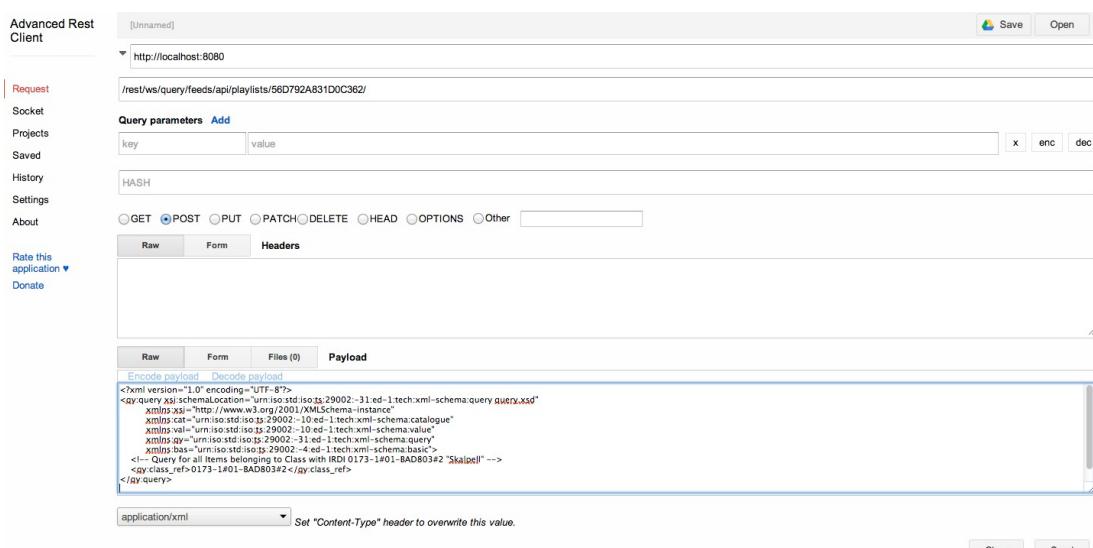


Abbildung E.1.: Advanced REST Client Test des Webservices

Anhang F.

PLIB Datenbanktabellen - Ausschnitt

Abbildung F.1 zeigt einen Ausschnitt der Datenbanktabellen. Die Abbildung wurde von Herrn Karsten Mende zur Verfügung gestellt.

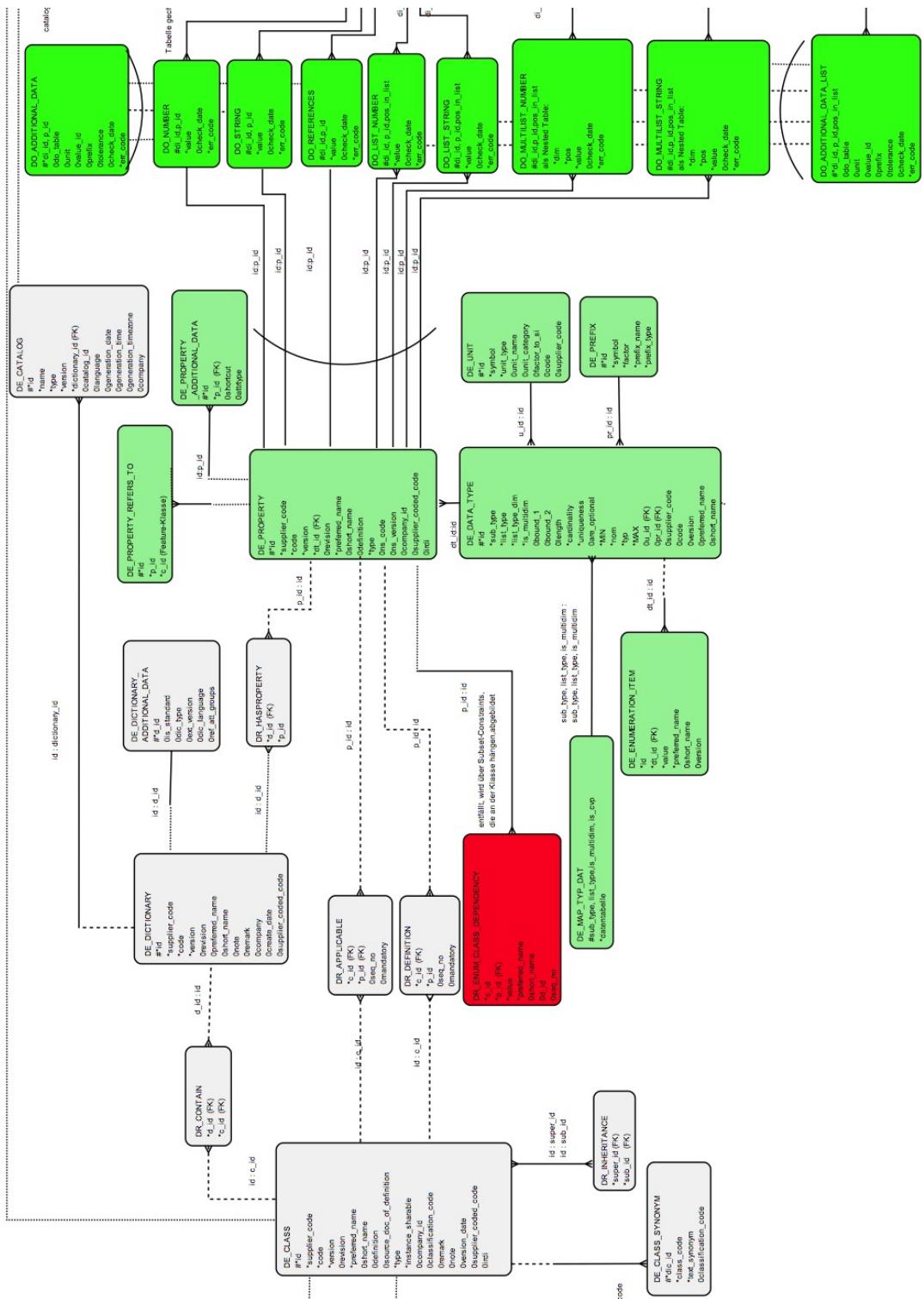


Abbildung F.1.: PLIB Datenbanktabellen - Ausschnitt

Anhang G.

SOAP Webservice

Um die beiden unterschiedlichen Webservicearten vergleichen zu können, wurde evaluiert, wie mit der gleichen Plattform, Programmiersprache und den gleichen Schemadateien ein SOAP Webservice implementiert werden kann.

Genutzt wurde die in der Java-JDK beinhaltete Implementierung der JAX-WS-2.1-Spezifikation, somit muss keine weitere Bibliothek eingebunden werden.

Die gesamte Business-Logik-Implementierung für den REST-Webservice kann genutzt werden. Für die Erstellung des Webservices sind folgende Schritte notwendig:

- Erstellung einer Webservice-Klasse, welche die Business Logik aufruft
- Erstellung eines Webservice-Servers, welche den Webservice bereitstellt

G.1. Erstellung einer Webservice-Klasse

Das Listing G.1 zeigt die Implementierung einer Webservice-Klasse.

Hierbei definiert @WebService diese Klasse mittels Annotation als Webservice gemäß JAX-WS Spezifikation.

@SOAPBinding(style=SOAPBinding.Style.DOCUMENT) definiert ein SOAP-Binding als Dokumenten Typ. Ein weiterer Typ ist RPC. @WebMethod definiert die markierte Methode als Webservice-Methode.

Die Methode leitet den übergebenen query einfach an die Business Logik, der QueryPipe, zur weiteren Verarbeitung. Dies entspricht exakt dem Vorgehen in der query-Methode im REST-Webservice.

Hinweis: In allen Quelltextbeispielen wird auf import- und package-Anweisungen verzichtet, um die Lesbarkeit zu erhöhen. Die Quelltexte liegen der Arbeit bei.

```
1  /**
2   * SOAP Webservice for querying via ISO 29002-31 Schema
3   */
4  @WebService
5  @SOAPBinding(style=SOAPBinding.Style.DOCUMENT)
6  public class QuerySOAPService {
7
8      /** Query processor instance */
9      private QueryPipe queryPipe;
10
11     /** Application context for receiving beans */
```

```
12     private ApplicationContext context;
13
14     public QuerySOAPService() {
15         this.context = initializeContext();
16         this.queryPipe = getQueryPipe();
17     }
18
19     /**
20      * Takes the query model and passes it to the query filter for further
21      * processing.
22      * Retreives the result and returns it.
23      * @param query the query
24      * @return the catalogue response
25      */
26     @WebMethod
27     public CatalogueType query(QueryType query) {
28         CatalogueType catalogue = queryPipe.filter(query);
29         return catalogue;
30     }
31
32     private ClassPathXmlApplicationContext initializeContext() {
33         return new ClassPathXmlApplicationContext(
34             "beans.xml");
35     }
36
37     private QueryPipe getQueryPipe() {
38         QueryPipe queryPipe = (QueryPipe) context.getBean("queryPipe");
39         return queryPipe;
40     }
```

Listing G.1: SOAP Webservice Klasse zum Senden eines Queries

G.2. Erstellung eines Webservice Servers

Das Listing G.2 zeigt die Implementierung eines beispielhaften Webservice-Servers. Dieser erzeugt eine Instanz der Klasse QuerySOAPService, der Klasse, welche die Methoden für den Webservice beinhaltet. Anschließend wird ein Endpoint definiert. Hier wird die spätere URL, unter dem der Webservice zur Verfügung gestellt wird, definiert und der QuerySOAPService übergeben.

Ferner wird zu Testzwecken, zum Stoppen des Services, ein Dialog-Fenster angezeigt.

```
1 package de.feuplib.webservice;
2
3 import org.apache.log4j.Logger;
4
5 import javax.swing.*;
6 import javax.xml.ws.Endpoint;
7
8 /**
9  * Is responsible for creating the SOAP Webserver.
10 */
11 public class QueryServer {
12     /** Logger instance */
13     static final Logger LOGGER = Logger.getLogger(StringServer.class);
```

```

14
15     public static void main(String args[]) {
16         LOGGER.info("Start query web service");
17         QuerySOAPService queryService = new QuerySOAPService();
18         Endpoint endpoint = Endpoint.publish(
19             "http://localhost:8088/soap/query", queryService);
20         LOGGER.info("Web service started");
21         JOptionPane.showMessageDialog(null, "Server beenden");
22         endpoint.stop();
23         LOGGER.info("Web service stopped");
24     }
25 }
```

Listing G.2: SOAP Webservice Server

G.3. SOAP Beispielrequest und -response

Das Listing G.3 zeigt einen Beispielrequest und Listing G.4 zeigt die entsprechende Antwort des Servers.

```

1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
2   xmlns:web="http://webserviceplib.feuerwehr.de/" xmlns:urn=
3     urn:iso:std:iso:ts:29002: -31:ed-1:tech:xml -schema:query" xmlns:urn1=
4     urn:iso:std:iso:ts:29002: -10:ed-1:tech:xml -schema:catalogue" xmlns:urn2=
5     urn:iso:std:iso:ts:29002: -4:ed-1:tech:xml -schema:basic" xmlns:urn3=
6     urn:iso:std:iso:ts:29002: -10:ed-1:tech:xml -schema:value">
7   <soapenv:Header/>
8   <soapenv:Body>
9     <web:query>
10    <arg0>
11      <urn:class_ref>0173-1#01-BAD803#2</urn:class_ref>
12    </arg0>
13    </web:query>
14  </soapenv:Body>
15 </soapenv:Envelope>
```

Listing G.3: Beispielanfrage SOAP Webservice query

```

1 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
2   <S:Body>
3     <ns6:queryResponse xmlns:ns2="urn:iso:std:iso:ts:29002: -10:ed-1:tech:xml -
4       schema:catalogue" xmlns:ns3="urn:iso:std:iso:ts:29002: -4:ed-1:tech:xml -
5       schema:basic" xmlns:ns4="urn:iso:std:iso:ts:29002: -10:ed-1:tech:xml -
6       schema:value" xmlns:ns5="urn:iso:std:iso:ts:29002: -31:ed-1:tech:xml -
7       schema:query" xmlns:ns6="http://webserviceplib.feuerwehr.de/">
8     <return>
9       <ns2:item class_ref="0173-1#01-BAD803#2">
10        <ns2:property_value property_ref="0173-1#02-AAA762#1">
11          <ns4:measure_single_number_value>
12            <ns4:real_value>2.0</ns4:real_value>
13          </ns4:measure_single_number_value>
14        </ns2:property_value>
15        <ns2:property_value property_ref="0173-1#02-AAB011#1">
16          <ns4:measure_single_number_value>
17            <ns4:real_value>1.0</ns4:real_value>
18          </ns4:measure_single_number_value>
19        </ns2:property_value>
20      </ns2:item>
21    </return>
22  </S:Body>
23 </S:Envelope>
```

```

16      </ns2:item>
17      <ns2:item class_ref="0173-1#01-BAD803#2">
18          <ns2:property_value property_ref="0173-1#02-AAA762#1">
19              <ns4:measure_single_number_value>
20                  <ns4:real_value>2.0</ns4:real_value>
21              </ns4:measure_single_number_value>
22          </ns2:property_value>
23          <ns2:property_value property_ref="0173-1#02-AAB011#1">
24              <ns4:measure_single_number_value>
25                  <ns4:real_value>1.0</ns4:real_value>
26              </ns4:measure_single_number_value>
27          </ns2:property_value>
28      </ns2:item>
29  </return>
30  </ns6:queryResponse>
31 </S:Body>
32 </S:Envelope>
```

Listing G.4: Beispielantwort SOAP Webservice response

Die Listing G.5 zeigt die WSDL des Query Services. Diese wurde automatisch generiert. Das geschieht durch den Aufruf der URL des Services mit Query-Parameter ?wsdl.

Folgende URL wurde aufgerufen: <http://localhost:8088/soap/query?wsdl>

Man sieht, dass die benötigten Schemadateien für query, basic, value und catalogue der ISO XSDs automatisch eingebunden werden. Der Grund ist, dass in den Java Sourcen wie in Listing G.1 ersichtlich, die bereits während der REST-Implementierung für das Marshalling und Unmarshalling erzeugten Zugriffsklassen mittels JAXB benutzt werden. Das sind CatalogueType und QueryType. Dies sind JAXB annotierte Klassen, somit werden automatisch die entsprechenden Schemadateien verwendet.

```

1  <!--
2  Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI
   2.1.6 in JDK 6.
3  -->
4  <!--
5  Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI
   2.1.6 in JDK 6.
6  -->
7 <definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://
   webserviceplib.feu.de/"
8     xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.
   xmlsoap.org/wsdl/"
9     targetNamespace="http://webserviceplib.feu.de/" name=
       "QuerySOAPServiceService">
10    <types>
11        <xsd:schema>
12            <xsd:import namespace="urn:iso:std:iso:ts:29002:-4:ed-1:tech:xml-
               schema:basic"
               schemaLocation="http://localhost:8088/soap/query?xsd=1"/>
13        </xsd:schema>
14        <xsd:schema>
15            <xsd:import namespace="urn:iso:std:iso:ts:29002:-31:ed-1:tech:xml-
               schema:query"
```

```

17                     schemaLocation="http://localhost:8088/soap/query?xsd=2" />
18     </xsd:schema>
19     <xsd:schema>
20         <xsd:import namespace="urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-
21             schema:value"
22                 schemaLocation="http://localhost:8088/soap/query?xsd=3" />
23     </xsd:schema>
24     <xsd:schema>
25         <xsd:import namespace="urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-
26             schema:catalogue"
27                 schemaLocation="http://localhost:8088/soap/query?xsd=4" />
28     </xsd:schema>
29     <xsd:schema>
30         <xsd:import namespace="http://webserviceplib.feu.de/"
31             schemaLocation="http://localhost:8088/soap/query?xsd=5" />
32     </xsd:schema>
33 </types>
34 <message name="query">
35     <part name="parameters" element="tns:query"/>
36 </message>
37 <message name="queryResponse">
38     <part name="parameters" element="tns:queryResponse"/>
39 </message>
40 <portType name="QuerySOAPService">
41     <operation name="query">
42         <input message="tns:query"/>
43         <output message="tns:queryResponse"/>
44     </operation>
45 </portType>
46 <binding name="QuerySOAPServicePortBinding" type="tns:QuerySOAPService">
47     <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="

48         document"/>
49     <operation name="query">
50         <soap:operation soapAction="" />
51         <input>
52             <soap:body use="literal" />
53         </input>
54         <output>
55             <soap:body use="literal" />
56         </output>
57     </operation>
58 </binding>
59 <service name="QuerySOAPServiceService">
60     <port name="QuerySOAPServicePort" binding="tns:QuerySOAPServicePortBinding
61         ">
62             <soap:address location="http://localhost:8088/soap/query" />
63         </port>
64     </service>
65 </definitions>

```

Listing G.5: WSDL des Query Services mit SOAP

G.4. Fazit

Da sämtliche Voraussetzungen bereits während der Implementierung des REST-Webservices in der Business Logik vorhanden waren, konnte in kurzer Zeit ein SOAP-Webservice

erstellt werden, der die gleichen Query-Funktionalitäten zur Verfügung stellt wie der REST-Webservice. Das Generieren des XML-Modells in Java-Klassen wurde bereits mittels JAXB durchgeführt, sodass diese Modelle erneut benutzt werden können. Java kennt dann die Modelle und die Bindung an die entsprechenden XML-Schema-Dateien, sodass diese für den Webservice zur Verfügung gestellt werden können. Es ist also ersichtlich, dass die Webservice Schnittstelle mit relativ geringem Aufwand ausgetauscht werden kann, wenn die Business Logik gekapselt wurde.

Anhang H.

Architektur - Bausteinsichten

Dieses Kapitel enthält weitere verfeinerte Bausteinansichten über das System.

H.1. Level 2 - Whiteboxansicht - Komponente XMLData

Die Komponente XMLData beinhaltet alle Datenmodelle für die beiden Hauptkonzepte »Query for characteristic data« nach. Siehe dazu Abbildung H.1.

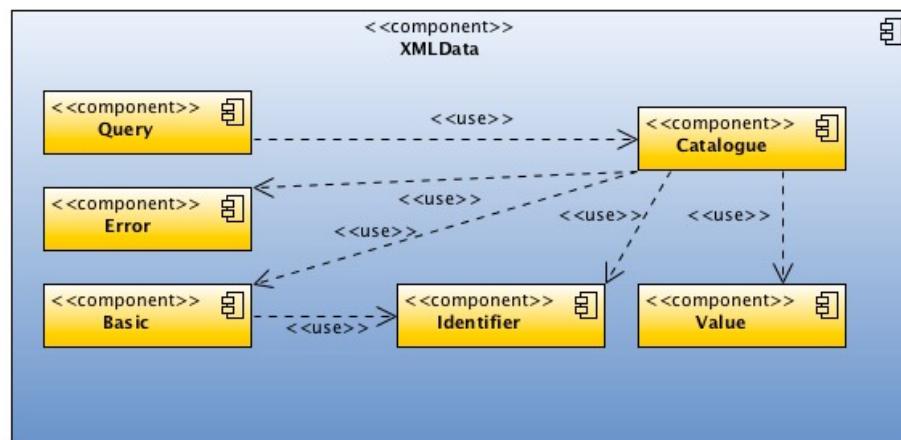


Abbildung H.1.: Bausteinsicht - Level 2 - Komponente XMLData

Query Diese Komponente beinhaltet das Datenmodell des Queries nach ISO/TS 29002-31.

Catalogue Diese Komponente beinhaltet das Datenmodell des Kataloges nach ISO/TS 29002-10.

Basic Diese Komponente beinhaltet das Datenmodell von Basistypen nach ISO/TS 29002-4.

Value Diese Komponente beinhaltet das Datenmodell der Wertetypen nach ISO/TS 29002-10.

Identifier Diese Komponente beinhaltet das Datenmodell für Identifier (IRDI) nach ISO/TS 29002-5.

H.2. Level 2 - Whiteboxansicht - Komponente Processor

Die Komponente Processor beinhaltet den QueryProcessor sowie die Handler und Analyser-Komponente. Siehe dazu Abbildung H.2.

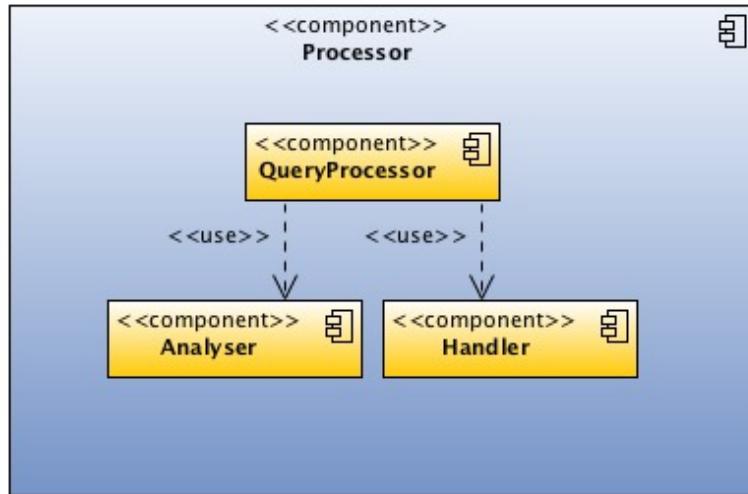


Abbildung H.2.: Bausteinsicht - Level 2 - Komponente Processor

QueryProcessor Diese Komponente ist für die erste Queryverarbeitung verantwortlich. Übernimmt eine Erkennung des Queries und leitet an den Handler und Analyser weiter.

Handler Diese Komponente beinhaltet die QueryServices, welche die eigentliche Verarbeitung und Transformation der Queries in die Datenmodelle übernimmt.

Analyser Enthält Komponenten zur Analyse und Markierung des Queries für die weitere Verarbeitung.

H.3. Level 3 - Whiteboxansicht - Komponente Handler

Die Komponente Handler beinhaltet die QueryServices, die zur Verarbeitung sowie Transformation der Queries zuständig sind. Siehe dazu Abbildung H.3.

SimpleQueryService Diese Komponente ist für die Verarbeitung und Transformation der SimpleQueries (Projektion) zuständig. Bereitet die Anfrage für die Datenbank für die PLIBDao-Komponente vor.

ParametricQueryService Diese Komponente ist für die Verarbeitung und Transformation der ParametricQueries (Selektion) zuständig. Bereitet die Anfrage für die Datenbank für die PLIBDao-Komponente vor.

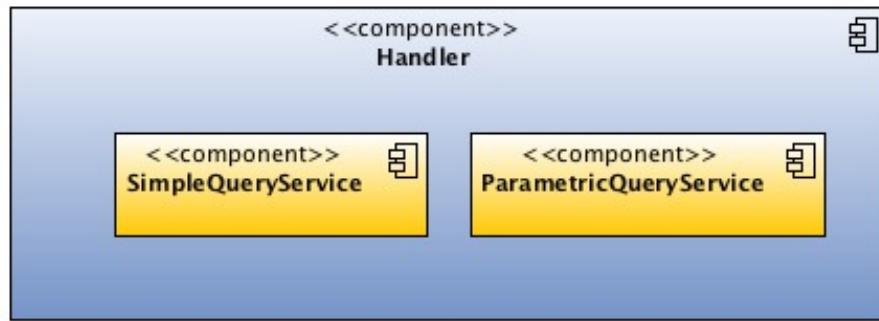


Abbildung H.3.: Bausteinsicht - Level 3 - Komponente Handler

H.4. Level 3 - Whiteboxansicht - Komponente Analyser

Die Komponente Analyser beinhaltet Komponenten zur Analyse und Markieren der Eingangsqueries. Siehe dazu Abbildung H.4.

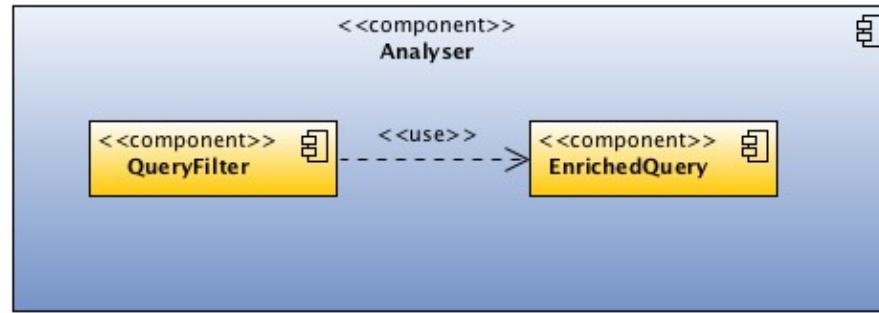


Abbildung H.4.: Bausteinsicht - Level 3 - Komponente Analyser

QueryFilter Diese Komponente filtert den Eingangsquery und markiert ihn mit Hilfe von EnrichedQuery.

EnrichedQuery Diese Komponente markiert den Query gemäß seiner Art (Simple, Parametric) zur späteren schnellen Erkennung. Es muss somit keine aufwändige Erkennung durch Prüfen der Eingangsdaten durchgeführt werden, sondern es können Flags in dieser Komponente abgefragt werden.

Anhang I.

Schema Dateien

Dieses Kapitel listet die Original Schema-Dateien der ISO-29002-31 auf. Diese wurden von Dr. Gerry Radack von der ECCMA zur Verfügung gestellt und wurden als Basis für die Implementierung verwendet.

I.1. query.xsd

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!--
3  $Id: query.xsd 411 2009-07-21 01:57:47Z radack $
4  ISO TC 184/SC 4/WG 12 N6664 - ISO/TS 29002-31 Query - XML schema
5  -->
6  <!--
7  The following permission notice and disclaimer shall be included in all copies of
     this XML schema ("the Schema"), and derivations of the Schema:
8
9  Permission is hereby granted, free of charge in perpetuity, to any person
    obtaining a copy of the Schema, to use, copy, modify, merge and distribute
    free of charge, copies of the Schema for the purposes of developing,
    implementing, installing and using software based on the Schema, and to
    permit persons to whom the Schema is furnished to do so, subject to the
    following conditions:
10
11 THE SCHEMA IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
    INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
    PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
    IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
    CONNECTION WITH THE SCHEMA OR THE USE OR OTHER DEALINGS IN THE SCHEMA.
12
13 In addition, any modified copy of the Schema shall include the following notice:
14
15 THIS SCHEMA HAS BEEN MODIFIED FROM THE SCHEMA DEFINED IN ISO 29002-31, AND SHOULD
    NOT BE INTERPRETED AS COMPLYING WITH THAT STANDARD.
16
17 -->
18 <xsschema xmlns:xss="http://www.w3.org/2001/XMLSchema"
19         xmlns:qy="urn:iso:std:iso:ts:29002:-31:ed-1:tech:xml-schema:query"
20         xmlns:cat="urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:catalogue"
21         xmlns:val="urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:value"
22         xmlns:bas="urn:iso:std:iso:ts:29002:-4:ed-1:tech:xml-schema:basic"
23         xmlns:id="urn:iso:std:iso:ts:29002:-5:ed-1:tech:xml-schema:identifier"
24         targetNamespace="urn:iso:std:iso:ts:29002:-31:ed-1:tech:xml-
            schema:query" elementFormDefault="qualified">
25   <xss:import namespace="urn:iso:std:iso:ts:29002:-4:ed-1:tech:xml-schema:basic"
26       schemaLocation="basic.xsd"/>
27   <xss:import namespace="urn:iso:std:iso:ts:29002:-5:ed-1:tech:xml-
            schema:identifier" schemaLocation="identifier.xsd"/>
```

```

27   <xs:import namespace="urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-
28     schema:catalogue" schemaLocation="catalogue.xsd"/>
29
30   <xs:import namespace="urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:value"
31     schemaLocation="value.xsd"/>
32
33   <!-- Global Elements -->
34   <xs:element name="characteristic_data_query_expression" type="
35     qy:characteristic_data_query_expression_Type"/>
36
37   <xs:element name="query" type="qy:query_Type"/>
38   <xs:element name="query_context" type="qy:query_context_Type"/>
39
40   <!-- Global Types -->
41
42   <!-- Characteristic Data Query Expression -->
43   <xs:complexType name="characteristic_data_query_expression_Type">
44     <xs:choice>
45       <xs:element name="cardinality" type="qy:cardinality_expression_Type"/>
46       <xs:element name="range" type="qy:range_expression_Type"/>
47       <xs:element name="string_pattern" type="
48         qy:string_pattern_expression_Type"/>
49       <xs:element name="string_size" type="qy:string_size_expression_Type"/>
50       <xs:element name="subset" type="qy:subset_expression_Type"/>
51       <xs:element name="data_environment" type="
52         qy:data_environment_expression_Type"/>
53       <xs:element name="or" type="qy:or_expression_Type"/>
54       <xs:element name="and" type="qy:and_expression_Type"/>
55       <xs:element name="not" type="qy:not_expression_Type"/>
56     </xs:choice>
57   </xs:complexType>
58
59   <!-- Or Expression -->
60   <xs:complexType name="or_expression_Type">
61     <xs:sequence>
62       <xs:element name="operand" type="
63         qy:characteristic_data_query_expression_Type" minOccurs="2"
64           maxOccurs="unbounded"/>
65     </xs:sequence>
66   </xs:complexType>
67
68   <!-- And query expression -->
69   <xs:complexType name="and_expression_Type">
70     <xs:sequence>
71       <xs:element name="operand" type="
72         qy:characteristic_data_query_expression_Type" minOccurs="2"
73           maxOccurs="unbounded"/>
74     </xs:sequence>
75   </xs:complexType>
76
77   <!-- Not Expression -->
78   <xs:complexType name="not_expression_Type">
79     <xs:sequence>
80       <xs:element name="operand" type="
81         qy:characteristic_data_query_expression_Type"/>
82     </xs:sequence>
83   </xs:complexType>
84
85   <!-- -->
86
87   <!-- Query Expression -->
88   <xs:complexType name="query_expression_Type" abstract="true">
89     <xs:sequence>
90       <xs:element name="property_reference" type="qy:property_reference_Type"
91         "/>
92     </xs:sequence>
93   </xs:complexType>
94
95   <!-- Cardinality Expression -->
96   <xs:complexType name="cardinality_expression_Type">
97     <xs:complexContent>
98       <xs:extension base="qy:query_expression_Type">
```

```

79          <xs:sequence>
80              <xs:element name="minimum" type="xs:int" minOccurs="0"/>
81              <xs:element name="maximum" type="xs:int" minOccurs="0"/>
82          </xs:sequence>
83      </xs:extension>
84  </xs:complexContent>
85 </xs:complexType>
86 <!-- Range Expression -->
87 <xs:complexType name="range_expression_Type">
88     <xs:complexContent>
89         <xs:extension base="qy:query_expression_Type">
90             <xs:sequence>
91                 <xs:element name="min_value" type="xs:float" minOccurs="0"/>
92                 <xs:element name="max_value" type="xs:float" minOccurs="0"/>
93                 <xs:element name="UOM_ref" type="id:IRDI_type" minOccurs="0"/>
94                 <xs:element name="UOM_code" type="xs:string" minOccurs="0"/>
95                 <xs:element name="currency_ref" type="id:IRDI_type" minOccurs=
96                     "0"/>
97                 <xs:element name="currency_code" type="
98                     bas:ISO_currency_code_Type" minOccurs="0"/>
99                 <xs:element name="is_inclusive" type="xs:boolean"/>
100            </xs:sequence>
101        </xs:extension>
102    </xs:complexContent>
103 <!-- String Pattern Expression -->
104 <xs:complexType name="string_pattern_expression_Type">
105     <xs:complexContent>
106         <xs:extension base="qy:query_expression_Type">
107             <xs:sequence>
108                 <xs:element name="pattern" type="xs:string"/>
109                 <xs:element name="language_ref" type="id:IRDI_type" minOccurs=
110                     "0"/>
111                 <xs:element name="language_code" type="
112                     bas:ISO_language_code_Type" minOccurs="0"/>
113                 <xs:element name="country_code" type="
114                     bas:ISO_country_code_Type" minOccurs="0"/>
115             </xs:sequence>
116         </xs:extension>
117     </xs:complexContent>
118 <!-- String Size Expression -->
119 <xs:complexType name="string_size_expression_Type">
120     <xs:complexContent>
121         <xs:extension base="qy:query_expression_Type">
122             <xs:sequence>
123                 <xs:element name="min_length" type="xs:int" minOccurs="0"/>
124                 <xs:element name="max_length" type="xs:int" minOccurs="0"/>
125                 <xs:element name="language_ref" type="id:IRDI_type" minOccurs=
126                     "0"/>
127                 <xs:element name="language_code" type="
128                     bas:ISO_language_code_Type" minOccurs="0"/>
129                 <xs:element name="country_code" type="
130                     bas:ISO_country_code_Type" minOccurs="0"/>
131             </xs:sequence>
132         </xs:extension>
133     </xs:complexContent>
134 <!-- Subset Expression -->
135 <xs:complexType name="subset_expression_Type">
136     <xs:complexContent>

```

```

132         <xs:extension base="qy:query_expression_Type">
133             <xs:sequence>
134                 <xs:element name="value" type="qy:value_Type" minOccurs="0"
135                     maxOccurs="unbounded"/>
136             </xs:sequence>
137         </xs:extension>
138     </xs:complexContent>
139     <!-- Value -->
140     <xs:complexType name="value_Type">
141         <xs:sequence>
142             <xs:group ref="val:value_Group"/>
143         </xs:sequence>
144     </xs:complexType>
145     <!-- Data Environment Expression -->
146     <xs:complexType name="data_environment_expression_Type">
147         <xs:complexContent>
148             <xs:extension base="qy:query_expression_Type">
149                 <xs:sequence>
150                     <xs:element name="environment" type="
151                         qy:characteristic_data_query_expression_Type"
152                         maxOccurs="unbounded"/>
153                 </xs:sequence>
154             </xs:extension>
155         </xs:complexContent>
156     </xs:complexType>
157     <!-- -->
158     <!-- Property Reference -->
159     <xs:complexType name="property_reference_Type">
160         <xs:sequence>
161             <xs:element name="index" type="xs:int" minOccurs="0"/>
162             <xs:element name="property" type="qy:property_reference_Type"
163                 minOccurs="0"/>
164         </xs:sequence>
165         <xs:attribute name="property_ref" type="id:IRDI_type" use="required"/>
166     </xs:complexType>
167     <!-- Query -->
168     <xs:complexType name="query_Type">
169         <xs:sequence>
170             <xs:element name="item_description" type="xs:string" minOccurs="0"/>
171             <xs:element name="data_specification_ref" type="id:IRDI_type"
172                 minOccurs="0"/>
173             <xs:element name="class_ref" type="id:IRDI_type" minOccurs="0"/>
174             <xs:element name="property_ref" type="id:IRDI_list_type" minOccurs="0"
175                 />
176             <xs:element ref="cat:item" minOccurs="0"/>
177             <xs:element ref="qy:characteristic_data_query_expression" minOccurs="0"
178                 maxOccurs="unbounded"/>
179         </xs:sequence>
180     </xs:complexType>
181     <!-- Query Context -->
182     <xs:complexType name="query_context_Type">
183         <xs:sequence>
184             <xs:element name="requesting_organization_ref" type="id:IRDI_type"/>
185             <xs:element name="request_date_time" type="xs:dateTime"/>
186             <xs:element name="response_due_date_time" type="xs:dateTime" minOccurs
187                 ="0"/>
188             <xs:element name="response_email_address" type="xs:string"/>
189             <xs:element name="point_of_contact" type="xs:string" minOccurs="0"/>
190             <xs:element ref="qy:query" minOccurs="0" maxOccurs="unbounded"/>
191         </xs:sequence>

```

```

186     </xs:complexType>
187 </xs:schema>
```

Listing I.1: query.xsd

I.2. catalogue.xsd

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!--
3  $Id: catalogue.xsd 332 2009-05-29 02:16:28Z radack $
4  ISO TC 184/SC 4/WG 12 N5178 - ISO/TS 29002-10 Catalogue - XML schema
5  -->
6  <!--
7  The following permission notice and disclaimer shall be included in all copies of
     this XML schema ("the Schema"), and derivations of the Schema:
8
9  Permission is hereby granted, free of charge in perpetuity, to any person
     obtaining a copy of the Schema, to use, copy, modify, merge and distribute
     free of charge, copies of the Schema for the purposes of developing,
     implementing, installing and using software based on the Schema, and to
     permit persons to whom the Schema is furnished to do so, subject to the
     following conditions:
10
11 THE SCHEMA IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
     INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
     PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
     COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
     IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
     CONNECTION WITH THE SCHEMA OR THE USE OR OTHER DEALINGS IN THE SCHEMA.
12
13 In addition, any modified copy of the Schema shall include the following notice:
14
15 THIS SCHEMA HAS BEEN MODIFIED FROM THE SCHEMA DEFINED IN ISO/TS 29002-10, AND
     SHOULD NOT BE INTERPRETED AS COMPLYING WITH THAT STANDARD.
16
17 -->
18 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:cat="
     urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:catalogue" xmlns:val="
     urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:value" xmlns:bas="
     urn:iso:std:iso:ts:29002:-4:ed-1:tech:xml-schema:basic" xmlns:id="
     urn:iso:std:iso:ts:29002:-5:ed-1:tech:xml-schema:identifier" targetNamespace="
     urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:catalogue"
     elementFormDefault="qualified">
19   <xs:import namespace="urn:iso:std:iso:ts:29002:-5:ed-1:tech:xml-
     schema:identifier" schemaLocation="identifier.xsd"/>
20   <xs:import namespace="urn:iso:std:iso:ts:29002:-4:ed-1:tech:xml-schema:basic"
     schemaLocation="basic.xsd"/>
21   <xs:import namespace="urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:value"
     schemaLocation="value.xsd"/>
22   <!-- Global Elements -->
23   <xs:element name="catalogue" type="cat:catalogue_Type"/>
24   <xs:element name="item" type="cat:item_Type"/>
25   <xs:element name="property_value" type="cat:property_value_Type"/>
26   <xs:element name="reference" type="cat:reference_Type"/>
27   <!-- Global Types -->
28   <!-- Catalogue -->
29   <xs:complexType name="catalogue_Type">
30     <xs:sequence>
31       <xs:element ref="cat:item" minOccurs="0" maxOccurs="unbounded"/>
```

```

32     </xs:sequence>
33 </xs:complexType>
34 <!-- Item -->
35 <xs:complexType name="item_Type">
36     <xs:sequence>
37         <xs:element name="classification_ref" type="id:IRDI_type" minOccurs="0"
38             maxOccurs="unbounded"/>
39         <xs:element ref="cat:reference" minOccurs="0" maxOccurs="unbounded"/>
40         <xs:element ref="cat:property_value" minOccurs="0" maxOccurs="unbounded"/>
41     </xs:sequence>
42     <xs:attribute name="class_ref" type="id:IRDI_type" use="required"/>
43     <xs:attribute name="data_specification_ref" type="id:IRDI_type" use="optional"
44         />
45     <xs:attribute name="local_id" type="xs:ID" use="optional"/>
46     <xs:attribute name="information_supplier_reference_string" type="xs:string"
47         use="optional"/>
48     <xs:attribute name="is_proprietary" type="xs:boolean" use="optional"/>
49     <xs:attribute name="is_dependent" type="xs:boolean" use="optional"/>
50     <xs:attribute name="is_global_id" type="xs:boolean" use="optional"/>
51     <xs:attribute name="is_model" type="xs:boolean" use="optional"/>
52     <xs:attribute name="created_view" type="id:IRDI_type" use="optional"/>
53     <xs:attribute name="view_of" type="xs:IDREF" use="optional"/>
54 </xs:complexType>
55 <!-- Property Value -->
56 <xs:complexType name="property_value_Type">
57     <xs:sequence>
58         <xs:group ref="val:extended_value_Group"/>
59         <xs:element ref="val:environment" minOccurs="0"/>
60     </xs:sequence>
61     <xs:attribute name="property_ref" type="id:IRDI_type" use="required"/>
62     <xs:attribute name="subitem_path_property_ref" type="id:IRDI_list_type" use="
63         optional"/>
64     <xs:attribute name="is_proprietary" type="xs:boolean" use="optional"/>
65 </xs:complexType>
66 <!-- Reference -->
67 <xs:complexType name="reference_Type">
68     <xs:sequence>
69         <xs:element name="designation" type="bas:international_text_Type" minOccurs=
70             "0"/>
71     </xs:sequence>
72     <xs:attribute name="organization_ref" type="id:IRDI_type" use="optional"/>
73     <xs:attribute name="organization_code" type="xs:string" use="optional"/>
74     <!-- Constraint: organization_ref or organization_code (or both) must be
75         specified. -->
76     <!-- Constraint: If both organization_ref and organization_code are specified,
77         they must denote the same organization. -->
78     <xs:attribute name="reference_number" type="xs:string" use="required"/>
79 </xs:complexType>
80 </xs:schema>

```

Listing I.2: catalogue.xsd

I.3. basic.xsd

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--
3 $Id: basic.xsd 411 2009-07-21 01:57:47Z radack $
4 ISO TC 184/SC 4/WG 12 N6650 - ISO/TS 29002-4 Basic - XML schema
5 -->

```

```

6  <!--
7  The following permission notice and disclaimer shall be included in all copies of
   this XML schema ("the Schema"), and derivations of the Schema:
8
9  Permission is hereby granted, free of charge in perpetuity, to any person
   obtaining a copy of the Schema, to use, copy, modify, merge and distribute
   free of charge, copies of the Schema for the purposes of developing,
   implementing, installing and using software based on the Schema, and to
   permit persons to whom the Schema is furnished to do so, subject to the
   following conditions:
10
11 THE SCHEMA IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
   INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
   PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
   COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
   IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
   CONNECTION WITH THE SCHEMA OR THE USE OR OTHER DEALINGS IN THE SCHEMA.
12
13 In addition, any modified copy of the Schema shall include the following notice:
14
15 THIS SCHEMA HAS BEEN MODIFIED FROM THE SCHEMA DEFINED IN ISO/TS 29002-4, AND
   SHOULD NOT BE INTERPRETED AS COMPLYING WITH THAT STANDARD.
16
17 -->
18 <xsschema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:basic=""
   urn:iso:std:iso:ts:29002:-4:ed-1:tech:xml-schema:basic" xmlns:id=""
   urn:iso:std:iso:ts:29002:-5:ed-1:tech:xml-schema:identifier" targetNamespace=""
   urn:iso:std:iso:ts:29002:-4:ed-1:tech:xml-schema:basic" elementFormDefault=""
   qualified">
19   <xss:import namespace="urn:iso:std:iso:ts:29002:-5:ed-1:tech:xml-
   schema:identifier" schemaLocation="identifier.xsd"/>
20   <!-- Global Types -->
21   <!-- Day Interval -->
22   <xssimpleType name="day_interval_Type">
23     <xss:union memberTypes="xs:gYear xs:gYearMonth xs:date"/>
24   </xssimpleType>
25   <!-- ASN.1 Identifier -->
26   <xssimpleType name="ASN1_identifier_Type">
27     <xss:restriction base="xs:string"/>
28   </xssimpleType>
29   <!-- International Text -->
30   <xsscomplexType name="international_text_Type">
31     <xss:sequence>
32       <xss:element name="local_string" type="basic:language_string_Type" maxOccurs=
         "unbounded"/>
33     </xss:sequence>
34   </xsscomplexType>
35   <!-- ISO Country Code -->
36   <xssimpleType name="ISO_country_code_Type">
37     <xss:restriction base="xs:string">
38       <xss:pattern value="[A-Z]{2}"/>
39     </xss:restriction>
40   </xssimpleType>
41   <!-- ISO Currency Code -->
42   <xssimpleType name="ISO_currency_code_Type">
43     <xss:restriction base="xs:string">
44       <xss:length value="3"/>
45     </xss:restriction>
46   </xssimpleType>
47   <!-- ISO Language Code -->
48   <xssimpleType name="ISO_language_code_Type">

```

```
49     <xs:restriction base="xs:string">
50         <xs:pattern value="[a-z]{2}" />
51         <xs:pattern value="[a-z]{3}" />
52     </xs:restriction>
53 </xs:simpleType>
54 <!-- Language String -->
55 <xs:complexType name="language_string_Type">
56     <xs:sequence>
57         <xs:element name="content" type="xs:string"/>
58         <xs:element name="language_ref" type="id:IRDI_type" minOccurs="0"/>
59         <xs:sequence minOccurs="0">
60             <xs:element name="language_code" type="basic:ISO_language_code_Type"/>
61             <xs:element name="country_code" type="basic:ISO_country_code_Type"
62                 minOccurs="0"/>
63         </xs:sequence>
64         <!-- Constraint: language or language_code (or both) must be specified. -->
65         <!-- Constraint: If both language and language_code are specified, then
66             language and the combination of language_code and country_code must
67             denote the same localized language. -->
68     </xs:sequence>
69 </xs:complexType>
70 </xs:schema>
```

Listing I.3: basic.xsd

I.4. identifier.xsd

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--
3 $Id: identifier.xsd 420 2009-07-25 21:51:57Z radack $
4 ISO TC 184/SC 4/WG 12 N5180 - ISO/TS 29002-5 Identifier - XML schema
5 -->
6 <!--
7 The following permission notice and disclaimer shall be included in all copies of
     this XML schema ("the Schema"), and derivations of the Schema:
8
9 Permission is hereby granted, free of charge in perpetuity, to any person
     obtaining a copy of the Schema, to use, copy, modify, merge and distribute
     free of charge, copies of the Schema for the purposes of developing,
     implementing, installing and using software based on the Schema, and to
     permit persons to whom the Schema is furnished to do so, subject to the
     following conditions:
10
11 THE SCHEMA IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
     INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
     PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
     COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
     IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
     CONNECTION WITH THE SCHEMA OR THE USE OR OTHER DEALINGS IN THE SCHEMA.
12
13 In addition, any modified copy of the Schema shall include the following notice:
14
15 THIS SCHEMA HAS BEEN MODIFIED FROM THE SCHEMA DEFINED IN ISO/TS 29002-5, AND
     SHOULD NOT BE INTERPRETED AS COMPLYING WITH THAT STANDARD.
16
17 -->
18 <!DOCTYPE xs:schema [
19     <!ENTITY % identifier.dtd SYSTEM "identifier.dtd">
20     %identifier.dtd;
```

```

21  ]>
22  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:id=
23    "urn:iso:std:iso:ts:29002:-5:ed-1:tech:xml-schema:identifier" targetNamespace=""
24    "urn:iso:std:iso:ts:29002:-5:ed-1:tech:xml-schema:identifier"
25    elementFormDefault="qualified" attributeFormDefault="unqualified">
26    <!-- IRDI -->
27    <xs:element name="IRDI" type="id:IRDI_type"/>
28    <xs:element name="IRDI_list" type="id:IRDI_list_type"/>
29    <xs:simpleType name="IRDI_type">
30      <xs:restriction base="xs:string">
31        <xs:pattern value="&irdi1;"/>
32        <xs:pattern value="&irdi2;"/>
33        <xs:pattern value="&irdi3;"/>
34      </xs:restriction>
35    </xs:simpleType>
36    <!-- IRDI sequence -->
37    <xs:complexType name="IRDI_sequence_type">
38      <xs:sequence>
39        <xs:element ref="id:IRDI" minOccurs="0" maxOccurs="unbounded"/>
40      </xs:sequence>
41    </xs:complexType>
42    <!-- IRDI list-->
43    <xs:simpleType name="IRDI_list_type">
44      <xs:list itemType="id:IRDI_type"/>
45    </xs:simpleType>
46  </xs:schema>

```

Listing I.4: identifier.xsd

I.5. value.xsd

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!--
3  $Id: value.xsd 332 2009-05-29 02:16:28Z radack $
4  ISO TC 184/SC 4/WG 12 N5192 - ISO/TS 29002-10 Value - XML schema
5  -->
6  <!--
7  The following permission notice and disclaimer shall be included in all copies of
     this XML schema ("the Schema"), and derivations of the Schema:
8
9  Permission is hereby granted, free of charge in perpetuity, to any person
     obtaining a copy of the Schema, to use, copy, modify, merge and distribute
     free of charge, copies of the Schema for the purposes of developing,
     implementing, installing and using software based on the Schema, and to
     permit persons to whom the Schema is furnished to do so, subject to the
     following conditions:
10
11 THE SCHEMA IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
     INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
     PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
     COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
     IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
     CONNECTION WITH THE SCHEMA OR THE USE OR OTHER DEALINGS IN THE SCHEMA.
12
13 In addition, any modified copy of the Schema shall include the following notice:
14
15 THIS SCHEMA HAS BEEN MODIFIED FROM THE SCHEMA DEFINED IN ISO/TS 29002-10, AND
     SHOULD NOT BE INTERPRETED AS COMPLYING WITH THAT STANDARD.
16

```

```

17 -->
18 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:val=""
19   urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:value" xmlns:bas=""
20   urn:iso:std:iso:ts:29002:-4:ed-1:tech:xml-schema:basic" xmlns:id="
21   urn:iso:std:iso:ts:29002:-5:ed-1:tech:xml-schema:identifier" targetNamespace=""
22   urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:value" elementFormDefault=""
23   qualified">
24   <xs:import namespace="urn:iso:std:iso:ts:29002:-5:ed-1:tech:xml-
25     schema:identifier" schemaLocation="identifier.xsd"/>
26   <xs:import namespace="urn:iso:std:iso:ts:29002:-4:ed-1:tech:xml-schema:basic"
27     schemaLocation="basic.xsd"/>
28   <!-- Global Elements -->
29   <xs:element name="combination" type="val:combination_Type"/>
30   <xs:element name="bag_value" type="val:bag_value_Type"/>
31   <xs:element name="boolean_value" type="val:boolean_value_Type"/>
32   <xs:element name="complex_value" type="val:complex_value_Type"/>
33   <xs:element name="composite_value" type="val:composite_value_Type"/>
34   <xs:element name="controlled_value" type="val:controlled_value_Type"/>
35   <xs:element name="currency_value" type="val:currency_value_Type"/>
36   <xs:element name="date_value" type="val:date_value_Type"/>
37   <xs:element name="date_time_value" type="val:date_time_value_Type"/>
38   <xs:element name="environment" type="val:environment_Type"/>
39   <xs:element name="file_value" type="val:file_value_Type"/>
40   <xs:element name="integer_value" type="val:integer_value_Type"/>
41   <xs:element name="item_reference_value" type="val:item_reference_value_Type"/>
42   <xs:element name="localized_text_value" type="val:localized_text_value_Type"/>
43   <xs:element name="measure_qualified_number_value" type="
44     val:measure_qualified_number_value_Type"/>
45   <xs:element name="measure_range_value" type="val:measure_range_value_Type"/>
46   <xs:element name="measure_single_number_value" type="
47     val:measure_single_number_value_Type"/>
48   <xs:element name="null_value" type="val:null_value_Type"/>
49   <xs:element name="one_of" type="val:one_of_Type"/>
50   <xs:element name="rational_value" type="val:rational_value_Type"/>
51   <xs:element name="real_value" type="val:real_value_Type"/>
52   <xs:element name="sequence_value" type="val:sequence_value_Type"/>
53   <xs:element name="set_value" type="val:set_value_Type"/>
54   <xs:element name="string_value" type="val:string_value_Type"/>
55   <xs:element name="time_value" type="val:time_value_Type"/>
56   <xs:element name="year_month_value" type="val:year_month_value_Type"/>
57   <xs:element name="year_value" type="val:year_value_Type"/>
58   <!-- Groups -->
59   <!-- Extended Value -->
60   <xs:group name="extended_value_Group">
61     <xs:choice>
62       <xs:group ref="val:value_Group"/>
63       <xs:element ref="val:one_of"/>
64       <xs:element ref="val:combination"/>
65     </xs:choice>
66   </xs:group>
67   <!-- Value -->
68   <xs:group name="value_Group">
69     <xs:choice>
70       <!-- The first element in the following list is the one that will be created
71           by default in some XML editors whenever a property_value is added. -->
72       <xs:element ref="val:string_value"/>
73       <xs:element ref="val:bag_value"/>
74       <xs:element ref="val:boolean_value"/>
75       <xs:element ref="val:complex_value"/>
76       <xs:element ref="val:composite_value"/>
77       <xs:element ref="val:controlled_value"/>

```

```

68      <xs:element ref="val:currency_value"/>
69      <xs:element ref="val:date_value"/>
70      <xs:element ref="val:date_time_value"/>
71      <xs:element ref="val:file_value"/>
72      <xs:element ref="val:integer_value"/>
73      <xs:element ref="val:item_reference_value"/>
74      <xs:element ref="val:localized_text_value"/>
75      <xs:element ref="val:measure_qualified_number_value"/>
76      <xs:element ref="val:measure_range_value"/>
77      <xs:element ref="val:measure_single_number_value"/>
78      <xs:element ref="val:null_value"/>
79      <xs:element ref="val:rational_value"/>
80      <xs:element ref="val:real_value"/>
81      <xs:element ref="val:sequence_value"/>
82      <xs:element ref="val:set_value"/>
83      <xs:element ref="val:time_value"/>
84      <xs:element ref="val:year_month_value"/>
85      <xs:element ref="val:year_value"/>
86    </xs:choice>
87  </xs:group>
88  <!-- Numeric Value -->
89  <xs:group name="numeric_value">
90    <xs:choice>
91      <!-- The first element in the following list is the one that will be created
         by default in some XML editors whenever a property_value is added. -->
92      <xs:element ref="val:real_value"/>
93      <xs:element ref="val:complex_value"/>
94      <xs:element ref="val:integer_value"/>
95      <xs:element ref="val:rational_value"/>
96    </xs:choice>
97  </xs:group>
98  <!-- Global Types -->
99  <!-- Bag Value -->
100 <xs:complexType name="bag_value_Type">
101   <xs:sequence>
102     <xs:group ref="val:value_Group" minOccurs="0" maxOccurs="unbounded"/>
103   </xs:sequence>
104 </xs:complexType>
105 <!-- Boolean Value -->
106 <xs:complexType name="boolean_value_Type">
107   <xs:simpleContent>
108     <xs:extension base="xs:boolean"/>
109   </xs:simpleContent>
110 </xs:complexType>
111 <!-- Combination -->
112 <xs:complexType name="combination_Type">
113   <xs:sequence>
114     <xs:group ref="val:value_Group" maxOccurs="unbounded"/>
115   </xs:sequence>
116 </xs:complexType>
117 <!-- Complex Value -->
118 <xs:complexType name="complex_value_Type">
119   <xs:sequence>
120     <xs:element name="real_part" type="xs:double"/>
121     <xs:element name="imaginary_part" type="xs:double"/>
122   </xs:sequence>
123 </xs:complexType>
124 <!-- Composite Value -->
125 <xs:complexType name="composite_value_Type">
126   <xs:sequence>

```

```

127      <xs:element name="field" type="val:field_Type" minOccurs="0" maxOccurs="unbounded"/>
128  
```

- 128 </xs:sequence>
- 129
 - 129 </xs:complexType>
 - 130
 - 130 <!-- Condition Element -->
 - 131
 - 131 <xs:complexType name="condition_element_Type">
 - 132
 - 132 <xs:sequence>
 - 133
 - 133 <xs:group ref="val:value_Group"/>
 - 134
 - 134 </xs:sequence>
 - 135
 - 135 <xs:attribute name="property_ref" type="id:IRDI_type" use="required"/>
 - 136
 - 136 </xs:complexType>
 - 137
 - 137 <!-- Controlled Value -->
 - 138
 - 138 <xs:complexType name="controlled_value_Type">
 - 139
 - 139 <xs:attribute name="value_ref" type="id:IRDI_type" use="optional"/>
 - 140
 - 140 <xs:attribute name="value_code" type="xs:string" use="optional"/>
 - 141
 - 141 <!-- Constraint: value_ref or value_code (or both) must be specified. -->
 - 142
 - 142 <!-- Constraint: If both value_ref and value_code specified, they must denote the same concept. -->
 - 143
 - 143 </xs:complexType>
 - 144
 - 144 <!-- Currency Value -->
 - 145
 - 145 <xs:complexType name="currency_value_Type">
 - 146
 - 146 <xs:sequence>
 - 147
 - 147 <xs:group ref="val:numeric_value"/>
 - 148
 - 148 </xs:sequence>
 - 149
 - 149 <xs:attribute name="currency_ref" type="id:IRDI_type" use="optional"/>
 - 150
 - 150 <xs:attribute name="currency_code" type="bas:ISO_currency_code_Type" use="optional"/>
 - 151
 - 151 <!-- Constraint: currency_ref or currency_code (or both) must be specified. -->
 - 152
 - 152 <!-- Constraint: If both currency_ref and currency_code are specified, they must denote the same concept. -->
 - 153
 - 153 </xs:complexType>
 - 154
 - 154 <!-- Date Value -->
 - 155
 - 155 <xs:complexType name="date_value_Type">
 - 156
 - 156 <xs:simpleContent>
 - 157
 - 157 <xs:extension base="xs:date"/>
 - 158
 - 158 </xs:simpleContent>
 - 159
 - 159 </xs:complexType>
 - 160
 - 160 <!-- Date Time Value -->
 - 161
 - 161 <xs:complexType name="date_time_value_Type">
 - 162
 - 162 <xs:simpleContent>
 - 163
 - 163 <xs:extension base="xs:dateTime"/>
 - 164
 - 164 </xs:simpleContent>
 - 165
 - 165 </xs:complexType>
 - 166
 - 166 <!-- Environment -->
 - 167
 - 167 <xs:complexType name="environment_Type">
 - 168
 - 168 <xs:sequence>
 - 169
 - 169 <xs:element name="property_value" type="val:condition_element_Type" maxOccurs="unbounded"/>
 - 170
 - 170 </xs:sequence>
 - 171
 - 171 </xs:complexType>
 - 172
 - 172 <!-- Field -->
 - 173
 - 173 <xs:complexType name="field_Type">
 - 174
 - 174 <xs:sequence>
 - 175
 - 175 <xs:group ref="val:value_Group"/>
 - 176
 - 176 </xs:sequence>
 - 177
 - 177 <xs:attribute name="property_ref" type="id:IRDI_type" use="optional"/>
 - 178
 - 178 </xs:complexType>
 - 179
 - 179 <!-- File Value -->
 - 180
 - 180 <xs:complexType name="file_value_Type">
 - 181
 - 181 <xs:sequence>

```

182      <xs:element name="URI" type="xs:anyURI"/>
183    </xs:sequence>
184  </xs:complexType>
185  <!-- Integer Value -->
186  <xs:complexType name="integer_value_Type">
187    <xs:simpleContent>
188      <xs:extension base="xs:int"/>
189    </xs:simpleContent>
190  </xs:complexType>
191  <!-- Item Reference Value -->
192  <xs:complexType name="item_reference_value_Type">
193    <xs:attribute name="item_local_ref" type="xs:IDREF" use="required"/>
194  </xs:complexType>
195  <!-- Localized Text Value -->
196  <xs:complexType name="localized_text_value_Type">
197    <xs:sequence>
198      <xs:element name="content" type="bas:international_text_Type"/>
199    </xs:sequence>
200  </xs:complexType>
201  <!-- Measure Qualified Number Value -->
202  <xs:complexType name="measure_qualified_number_value_Type">
203    <xs:complexContent>
204      <xs:extension base="val:measure_value_Type">
205        <xs:sequence>
206          <xs:element name="qualified_value" type="val:qualified_value_Type"
207            maxOccurs="unbounded"/>
208        </xs:sequence>
209      </xs:extension>
210    </xs:complexContent>
211  </xs:complexType>
212  <!-- Measure Range Value -->
213  <xs:complexType name="measure_range_value_Type">
214    <xs:complexContent>
215      <xs:extension base="val:measure_value_Type">
216        <xs:sequence>
217          <xs:element name="lower_value" type="val:numeric_value_Type"/>
218          <xs:element name="upper_value" type="val:numeric_value_Type"/>
219        </xs:sequence>
220      </xs:extension>
221    </xs:complexContent>
222  </xs:complexType>
223  <!-- Measure Single Number Value -->
224  <xs:complexType name="measure_single_number_value_Type">
225    <xs:complexContent>
226      <xs:extension base="val:measure_value_Type">
227        <xs:sequence>
228          <xs:choice>
229            <xs:group ref="val:numeric_value"/>
230          </xs:choice>
231        </xs:sequence>
232      </xs:extension>
233    </xs:complexContent>
234  </xs:complexType>
235  <!-- Measure Value -->
236  <xs:complexType name="measure_value_Type" abstract="true">
237    <xs:attribute name="UOM_ref" type="id:IRDI_type" use="optional"/>
238    <xs:attribute name="UOM_code" type="xs:string" use="optional"/>
239    <!-- Constraint: UOM_ref or UOM_code (or both) must be specified. -->
240    <!-- Constraint: If UOM_ref or UOM_code are specified, they must denote the
         same concept. -->
241  </xs:complexType>
```

```

241    <!-- Null Value -->
242    <xss:complexType name="null_value_Type"/>
243    <!-- Numeric Value -->
244    <xss:complexType name="numeric_value_Type">
245        <xss:sequence>
246            <xss:group ref="val:numeric_value"/>
247        </xss:sequence>
248    </xss:complexType>
249    <!-- OneOf -->
250    <xss:complexType name="one_of_Type">
251        <xss:sequence>
252            <xss:choice maxOccurs="unbounded">
253                <xss:element ref="val:combination"/>
254                <xss:group ref="val:value_Group"/>
255            </xss:choice>
256        </xss:sequence>
257    </xss:complexType>
258    <!-- Qualified Value -->
259    <xss:complexType name="qualified_value_Type">
260        <xss:sequence>
261            <xss:group ref="val:numeric_value"/>
262        </xss:sequence>
263        <xss:attribute name="qualifier_ref" type="id:IRDI_type" use="optional"/>
264        <xss:attribute name="qualifier_code" type="xs:string" use="optional"/>
265        <!-- Constraint: qualifier_ref or qualifier_code (or both) must be specified.
266            -->
267        <!-- Constraint: If both qualifier_ref or qualifier_code are specified, they
268            must denote the same concept. -->
269    </xss:complexType>
270    <!-- Rational Value -->
271    <xss:complexType name="rational_value_Type">
272        <xss:sequence>
273            <xss:element name="whole_part" type="xs:int" minOccurs="0"/>
274            <xss:element name="numerator" type="xs:int"/>
275            <xss:element name="denominator" type="xs:int"/>
276        </xss:sequence>
277    </xss:complexType>
278    <!-- Real Value -->
279    <xss:complexType name="real_value_Type">
280        <xss:simpleContent>
281            <xss:extension base="xs:double"/>
282        </xss:simpleContent>
283    </xss:complexType>
284    <!-- Sequence Value -->
285    <xss:complexType name="sequence_value_Type">
286        <xss:sequence>
287            <xss:group ref="val:value_Group" minOccurs="0" maxOccurs="unbounded"/>
288        </xss:sequence>
289    </xss:complexType>
290    <!-- Set Value -->
291    <xss:complexType name="set_value_Type">
292        <xss:sequence>
293            <xss:group ref="val:value_Group" minOccurs="0" maxOccurs="unbounded"/>
294        </xss:sequence>
295    </xss:complexType>
296    <!-- String Value -->
297    <xss:complexType name="string_value_Type">
298        <xss:simpleContent>
299            <xss:extension base="xs:string"/>

```

```
300  <!-- Time Value -->
301  <xss:complexType name="time_value_Type">
302      <xss:simpleContent>
303          <xss:extension base="xss:time"/>
304      </xss:simpleContent>
305  </xss:complexType>
306  <!-- Year Month Value -->
307  <xss:complexType name="year_month_value_Type">
308      <xss:simpleContent>
309          <xss:extension base="xss:gYearMonth"/>
310      </xss:simpleContent>
311  </xss:complexType>
312  <!-- Year Value -->
313  <xss:complexType name="year_value_Type">
314      <xss:simpleContent>
315          <xss:extension base="xss:gYear"/>
316      </xss:simpleContent>
317  </xss:complexType>
318 </xss:schema>
```

Listing I.5: value.xsd

Index

- Abfrage- und Antwortschnittstelle, 4
- Abgrenzung, 6, 7
- Advanced REST Client, 82
- Analyse und Definition der Anforderungen, 11
- Apache Tomcat, 21, 26
 - Applikationskontext, 29
- Applikationsserver, 23
- Aufgabenbeschreibung, 3
- Bibliotheken und Frameworks, 21
- Binding, 32
- Buildprozess, 33
- Characteristic query, 23
- Codegenerierung, 31
- Configuration Management, 25
- CURL, 82
- Datenbankserver, 23
- Datenflüsse, 8
- ss, 4
- Entwicklertest, 27
- Gesamtkontext, 6
- Geschäftsprozesse, 3
- GIT, 25
- HTTP-Methode
 - DELETE, 30
 - GET, 30
 - POST, 30
 - PUT, 30
- Identification Guide, 7, 63
- Implementierung, 25
- Integrationstest, 27, 74
- IRDI, 4, 38
- ISO 22745-30, 12, 63
- ISO 29002-10, 7, 8
- ISO 29002-31, 8, 11, 59
- JAXB, 31, 32
- JDBC, 45
- Jersey, 21, 28
- JSF2.0, 21
- JUnit, 71
- Klient, 22
- Kontextabgrenzung, 22
- Manuelle Entwicklertests, 27, 79
- Maven, 26, 33
 - pom.xml, 34
- Maven-Plugin, 34
- MIME-Type, 30
- Namespace, 32
- Object-Type, 46
- Ontologie, 4
- Oracle, 7
- Oracle Prozeduren, 35
- PLIB, 23
- PLIB-Prozeduren, 35
- Problemstellung, 3
- Programmiersprache, 22
- Projektion, 4
- Query, 30
 - Parametric Query, 63
 - Simple Query, 62
- Query Verarbeitung, 35
- Record-Types, 45
- REST, 20
 - RESTful, 19, 20
 - RESTful Webservice, 20, 28
- RESTful Webservice, 20
- Schema

basic.xsd, 100
catalogue.xsd, 99
identifier.xsd, 102
query.xsd, 95
value.xsd, 103
Schema-Dateien, 95
Selektion, 4
SOAP, 19
Software-Bausteine, 22
Softwarearchitektur, 22
Softwaredesign, 22
Spring, 22
 Data Oracle, 22
Spring Data Oracle, 41
SQL, 4
System- und Softwareentwurf, 19

Tests, 27
Testszenarios, 79

Unit Test, 27, 71
Use Cases, 7

VCS, 34
Versionierung, 34
Versionsverwaltung, 25
 verteilte Versionsverwaltung, 25
Vorgaben, 7

Web Container, 21
Web Server, 21
Webservice, 8, 19, 28
 SOAP, 85
Whiteboxansicht, 91–93
WSDL, 19

XSD, 31

Zielsetzung, 4