Qbox-Services:
Towards a Service-Oriented Quality Platform

September 2008 – January 2009

Laura González
Internship Report

Laboratoire PRiSM

Université de Versailles Saint-Quentin-en-Yvelines

This report summarizes the work I did during my internship at the Laboratorie PRiSM (Université de Versailles Saint-Quentin-en-Yvelines) from September 2008 to January 2009.

Content:

1 Introduction

QBox-Foundation [4] is a quality management platform which consists in a generic meta-model that allows users to define their quality goals, quality metrics and a set of appropriate measurement methods. However, some experiments revealed some limitations in this platform: incompleteness of quality methods and lack of connectivity to integrate functionalities of existing tools.

In addition, a large amount of tools has been implemented in order to deal with data quality in information systems. They support many types of data sources (relational databases, XML files, text files, etc) and provide different kinds of functionalities (measurement, analysis, improvement, etc). Datacleaner [5] and Talend Open Profiler [10] are some examples of such tools.

Service-oriented architecture (SOA) presents an approach for building distributed systems that deliver application functionality as services. SOAs provide many benefits to organizations like allowing them to leverage existing assets and composing new services out of existing ones. In this way SOAs help organizations to reduce cost and increase reuse.

In this report, a new approach for the QBox platform is presented which, taking advantage of service-oriented mechanisms, addresses the limitations of the QBox-Foundation platform.

The remaining of this report is organized as follows: section 2 presents an overview of Qbox-Foundation. In section 3 various open source data quality tools are described and compared. Section 4 describes service-oriented architecture and the Web Services technology as a way to implement it. Section 5 introduces the new service-oriented approach for the QBox platform. Section 6 presents the development of a prototype for this new approach. Finally, Section 7 presents conclusions and future work.

2 QBox Foundation Overview

Qbox-Foundation [4] is a metadata platform for quality assessment based on the Goal-Question-Metric (GQM) paradigm [1]. It proposes analyzing quality at three abstraction levels: (i) at conceptual level, identifying high-level quality goals, (ii) at operational level, enouncing a set of questions that characterize the way to assess each goal, and (iii) at quantitative level, defining a set of quality measures that quantify the way to answer to each question and a set of measurement methods for computing them.

2.1 General Description

The heart of Qbox-Foundation is a quality assessment meta-model, which allows understanding and reasoning with quality concepts. It is the result of successive refinements of the GQM paradigm, done in DWQ [2] and Quadris [3] projects. Figure 1gives a synthesized picture of this meta-model.

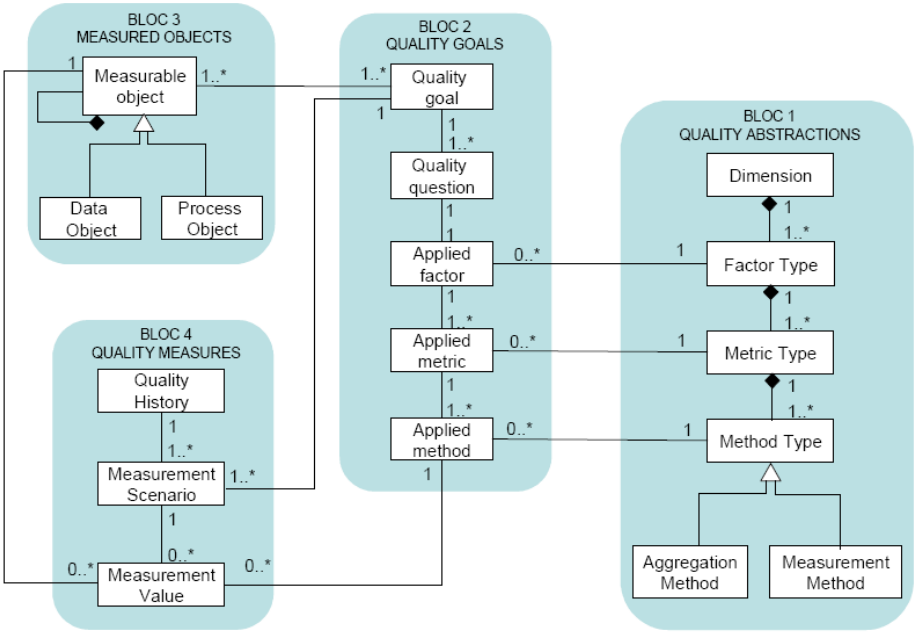

Figure 1 - Quality Assessment Metamodel

The central bloc of this quality meta-model (bloc 2) deals with quality goals following the GQM approach. Quality goals represent high-level quality needs (e.g. “reducing the number of returns in customer mails”). Quality questions represent the refinement/decomposition of a goal, reaching an operational form (an example of question for the previous goal is “which is the amount of syntactic errors in customer addresses?”). The answer to a quality question is defined by choosing and specializing a quality factor which best characterizes the question (e.g. syntactic correctness), a set of quality metrics which are appropriate to measure this factor (e.g. the percentage of data satisfying a syntax rule) and a set of methods of measurement of this metric. Quality factors, metrics and methods are chosen from a library of generic quality concepts (bloc 1 of the meta-model) and adapted to goal characteristics (e.g. checking for specific syntactic errors that commonly appear in address data).

The first bloc of the meta-model constitutes an extensible library of abstract data types which will be used to characterize specific quality goals. The main abstractions are: quality dimensions which capture a high-level facet of data quality, quality factors which represent particular aspects of quality dimensions, quality metrics which are the instruments used to measure quality factors and quality methods which are processes that implements a quality metric.

The third bloc of the meta-model refers to the information system (IS) model and to the processes which operate on the instances of this model. Each object type, being either a datum or a process, is called a measurable object if it is subject to a qualitative evaluation within a quality goal.

The fourth bloc of the meta-model deals with quality measurements. Given the definition of a quality goal, at any moment there will be a need to evaluate the quality questions and to analyze the obtained values in order to improve the quality of the measured objects. Measurement values represents the result of executing a measurement method (for evaluating a quality goal), for a measurable object, at a given instant or during a period of time. Results of successive quality measurements serve to analyze behaviors and trends of the measured objects. Generally, improvement actions are taken based on this analysis. A deeper description of the meta-model can be found in [4].

Qbox-Foundation was implemented as a Java web application, with user interfaces for managing the different entities of the meta-model and executing measurement methods. Its main functionalities include: management of an extensible catalog of quality concepts (dimensions, factors, metrics), management of an extensible library of methods (descriptions and code), definition and storage of user's quality goals and questions, association of quality goals to IS objects, selection and specialization of quality factors, selection and specialization of quality metrics, selection and binding of measurement methods, execution of measurement methods and visualization of measurement results.

2.2 QBox-Foundation Limitations

Even though various experiments have shown the relevance and the usefulness of the QBox-Foundation, they have also revealed some limitations of this first version of the QBox: incompleteness of quality methods and lack of connectivity to integrate functionalities of existing tools. These limitations can be seen as a problem of scalability, given that developing quality methods is time and money consuming, and although many tools in the market provide generic quality methods, they do not interoperate with each other and they do not share a common quality model.

3 Data Quality Tools

A large amount of tools has been implemented in order to deal with data quality problems in information systems. In this section, various open source data quality tools are described and compared according to, among others, the functionalities they provide and the data source types they support.

3.1 DataCleaner Project

DataCleaner [5] is an open source project, concerned with creating a data quality solution for business and organizations wishing to measure and increase the quality of their data. DataCleaner provides functionalities for profiling, validating and comparing data. Figure 2 shows the main user interface of the DataCleaner tool.

Figure 2 - DataCleaner User Interface

The profiling functionalities are grouped in profiles. Some examples of the built-in profiles and its functionalities are: standard measures profile (highest and lowest values, number of null values, row count), string analysis profile (word count, percentage of lowercase and uppercase chars, percentage of non letter chars) and number analysis profile (mean, geometric mean, standard deviation, sum, variance). Regarding validation, DataCleaner provides functionalities to perform validations based on dictionaries, regular expressions, range of values and JavaScript programs. It is important to note that although the tool provides a build-in number of functionalities, it also gives the mechanisms to incorporate new functionalities in the form of new profiles and validation rules. Figure 3 shows the results of running the standard measures profile in some columns of a table named “CUSTOMERS”.

Figure 3 - Standard Measures Results

The DataCleaner solution is structured in three main modules. The DataCleaner Core module is the core library of DataCleaner and includes classes and methods that provide a framework for data quality. The DataCleaner Gui module is a Swing based standalone desktop application which relies on the DataCleaner Core module to offer user oriented functionalities. Finally, the DataCleaner Monitor module, still in an early stage, is a Web based application for monitoring and scheduling continuous data quality efforts based on rules created in the desktop application. Additionally DataCleaner depends heavily on a component named MetaModel, which provides a common domain model for structures and querying of data stores.

Finally, in a more technical aspect, the DataCleaner tool is implemented with the Java programming language, so it is platform independent and can run on almost any operating system with a Java Runtime Environment (JRE). The tool supports many types of data sources like files in different formats (csv, xml, txt) and relational databases via JDBC. Table 1 presents a summary of the main DataCleaner characteristics.

Functionalities	Data Sources	License	Programming Language	Last version - Release Date
Calculus	JDBC, CSV, Excel, OpenOffice DBs, XML	Open Source – Apache License	Java	1.4 - 21/09/2008

Table 1 - Summary of DataCleaner characteristics

3.2 Aggregate Profiler

Aggregate Profiler [6] is an open source Data Quality and Data Profiling tool, which supports RDBMS (oracle, MySQL, MS Access and SQL Server), Flat Files, XML and XLS file formats for analysis. This tool can be used for profiling of data, quality check (and correction), and analysis of data (statistical analysis, charts). Additionally, it can perform cardinality checks between different tables within one data source. Finally, Aggregate Profiler can also be used for random generation of data, populating database values, looking into database metadata, fetching and storing data from/to databases.

Figure 4 shows the main user interface of the Aggregate Profiler tool and Table 2 presents a summary of its main characteristics.

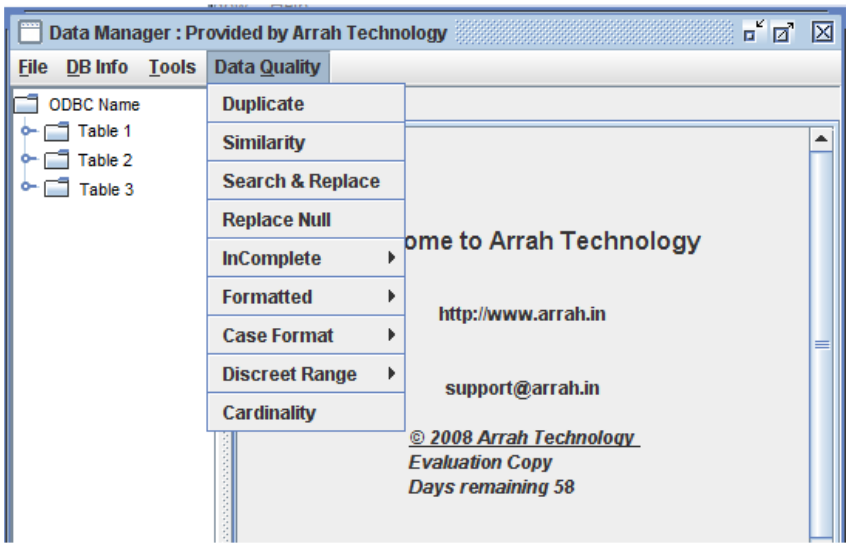

Figure 4 - Aggregate Profiler User Interface

Functionalities	Data Sources	License	Programming Language	Last version - Release Date
Calculus and Correction	Oracle, Access, MySQL and MS SQL (via ODBC)	Open Source - (GNU or LGPL)	Java	4.10 - 18/07/2008

Table 2 - Summary of Aggregate Profiler characteristics

3.3 Open Data Quality Project

The Open Data Quality [7] project is a Sub-component of Mural [8] community that provides an alternative to closed-source data management products. The project provides the capability to match, standardize, analyze, and cleanse data from various sources. It supports pluggable rules and standardization algorithms to allow support for new locales and vertical market applications. Additionally it supports pluggable matching algorithms for customized match logic and a high degree of flexibility through configuration. Figure 5 shows the main user interface of the Open Data Quality tool.

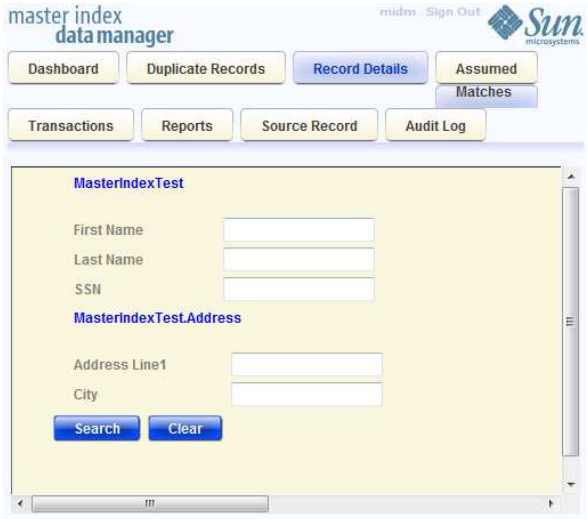

Figure 5 - Open Data Quality User Interface

Table 3 presents a summary of the main Open Data Quality tool characteristics.

Functionalities	Data Sources	License	Programming Language	Last version - Release Date
Code Generation (calculus and correction)	Various through Open Data Integrator Project [9]	Open Source - CDDL	Java (NetBeans plugins), Code generation in Java	r6u1 – 12/12/2008

Table 3 - Summary of Open Data Quality characteristics

3.4 Talend Open Profiler

Talend Open Profiler [10] is a data profiler that allows business users or data management staff to define a set of indicators for each data element that needs to be analyzed or monitored. The indicators that can be set with Talend Open Profiler include:

- **Simple statistics**
  They provide statistics on the number of records falling in certain categories, including the number of rows, the number of null values, the number distinct and unique values, the number of duplicates, or the number of blank fields.
- **Text statistics**
  They analyze the characteristics of text fields, including minimum, maximum and average length.
- **Summary statistics**
  They perform statistical analysis on numeric data, including the computation of the mean, the average, the inner quartile range, and the definition of ranges.
- **Advanced statistics**
  They determine the mode and builds frequency tables.

Additionally, Talend Open Profiler provides a user interface which presents a series of tables and graphs that display the results of the profiling for each data element and each indicator selected. Figure 6 shows and example of a Talend Open Profiler Graph and Table 4 presents a summary of the main Talend Open Profiler characteristics.

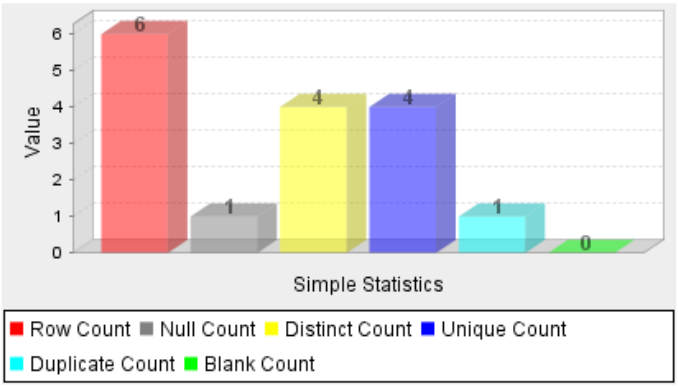

Figure 6 – Example of a Talend Open Profiler Graph

Functionalities	Data Sources	License	Programming Language	Last version - Release Date
Calculus	DB2, MySQL, Oracle, PostgreSQL, SQLServer. Probably more via JDBC.	Open Source - GPL	Java (Eclipse plugins)	1.1.4 – 27/02/2009

Table 4 - Summary of Talend Open Profiler characteristics

3.5 Power MatchMaker

Power Matchmaker is a tool created by data warehouse designers which contains many features geared for anyone dealing with information systems. For example, it provides functionalities to transform and cleanse data, validate and correct address information, accepts user-defined data matching criteria, merge duplicate records and their related data, builds cross-reference tables to link source systems' identifiers (Primary Keys) to the target database identifiers. Additionally, it provides an intuitive interface for match verification, which allows user confirmation of duplicates through the online verification facility. It also allows the Backup of impacted records prior to data merging and runs against the entire database to perform initial data cleanup, or incorporated into the data load process. Finally, the Power MatchMaker tool works with most database platforms.

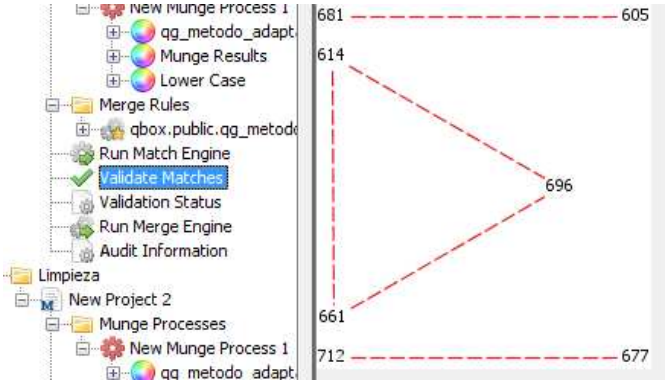

Figure 7 - Power MatchMaker User Interface

Functionalities	Data Sources	License	Programming Language	Last version - Release Date
Calculus and Correction	JDBC	Open Source - GPL	Java	0.9.4 – 08/08/2008

Table 5 - Summary of Power MatchMaker characteristics

3.6 Summary

Table 6 presents a summary of the characteristics of the tools reviewed in the previous sub-sections.

Tool	Functionalities	Data Sources	License	Programming Language	Last version - Release Date
Data Cleaner	Calculus	JDBC, CSV, Excel, OpenOffice DBs, XML	Open Source – Apache License	Java	1.4 - 21/09/2008
Aggregate Profiler	Calculus and Correction	Oracle, Access, MySQL and MS SQL (via ODBC)	Open Source - (GNU or LGPL)	Java	4.10 - 18/07/2008
Open Data Quality	Code Generation (calculus and correction)	Various through Open Data Integrator Project [9]	Open Source - CDDL	Java (NetBeans plugins), Code generation in Java	r6u1 – 12/12/2008
Talend Open Profiler	Calculus	DB2, MySQL, Oracle, PostgreSQL, SQLServer. Probably more via JDBC.	Open Source - GPL	Java (Eclipse plugins)	1.1.4 – 27/02/2009
Power Match Maker	Calculus and Correction	JDBC	Open Source - GPL	Java	0.9.4 – 08/08/2008

Table 6 - Summary of Data Quality Tools Characteristics

Although, there are nowadays many other commercial and open source data quality tools, the previous sections show that considering only a small amount of them, many heterogeneities arise. For example, the tools differ among each other in the types of data sources they support and the kinds of functionalities they provide. Additionally each of these tools is generally a stand-alone and user oriented application, which does not provide interoperation mechanisms with other tools. Furthermore, they manage different quality concepts, at different abstraction levels, expressed with ad-hoc terminology. These heterogeneities constitute a problem for the quality analyst who may require using functionalities provided by more than one tool according to specific needs.

4 SOA and Web Services

Service-oriented architecture (SOA) presents an approach for building distributed systems that deliver application functionality as services. Nowadays, Web Services constitute a well suitable technology for implementing a SOA. In this section, the SOA approach and the Web Service technology are described [12][13][14].

4.1 Service Oriented Architectures

SOA presents an approach for building distributed systems that deliver application functionality as services to either end-user applications or other services. The collaborations in a service-oriented architecture follow the "find, bind and invoke" paradigm where a service consumer performs dynamic service location by querying the service registry, according to specific requirements. If the service is found, the registry provides the consumer with the interface contract and the endpoint address for the service. Figure 8 presents the collaborations among the entities in a service-oriented architecture.

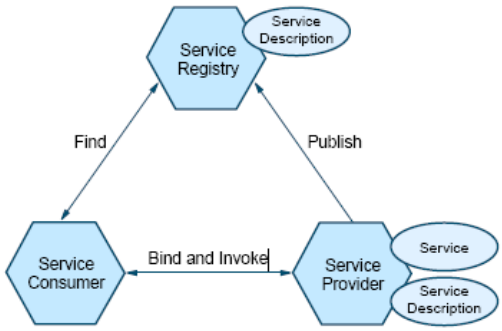

Figure 8 - Collaborations in a SOA

A service consumer is an application, a software module or another service that requires a service. It initiates the enquiry of the service in the registry, binds to the service over a transport, and executes the service function. The service consumer executes the service according to the interface contract. A service provider is a network-addressable entity that accepts and executes requests from consumers. It publishes its services and interface contract to the service registry so that the service consumer can discover and access the service. Finally, a service registry is the enabler for service discovery. It contains a repository of available services and allows for the lookup of service provider interfaces to interested service consumers.

In a SOA, services map to business functions. Each service has a well-defined interface that allows it to be published, discovered and invoked. An enterprise can choose to publish its services externally to business partners or internally within the organization. A service can also be composed from other services. Services are self-contained, modular and support interoperability. Additionally, they are loosely coupled and location-transparent.

SOAs can provide several benefits to help organizations to deal with their current two fundamental concerns: the ability to change quickly and the need to reduce costs.

First, SOAs allow organizations to leverage existing assets providing a layer of abstraction that enables an organization to continue leveraging its investment in IT by wrapping these existing assets as services that provide business functions. Within a SOA, the integration point is the service specification and not the implementation, providing in this way implementation transparency and minimizing the impact when infrastructure and implementation changes occur.

Additionally, SOAs provides a more responsive and faster time-to-market. The ability to compose new services out of existing ones provides a distinct advantage to an organization that has to be agile to respond to demanding business needs. This leads to rapid development of new business services and allows an organization to respond quickly to changes and reduce the time-to-market.

SOAs also help to reduce cost and increase reuse, given that with core business services exposed in a loosely coupled manner, they can be more easily used and combined based on business needs. This clearly means less duplication of resources, more potential for reuse, and lower costs.

4.2 Web Services

The W3C's Web Services Architecture Working Group [21] defines a Web service as a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols. Basic Web services combine the power of two ubiquitous technologies: XML, the universal data description language; and the HTTP transport protocol widely supported by browser and Web servers.

The Web Service technology is based on three fundamental standards: Simple Object Access Protocol (SOAP) [15], Web Service Description Language (WSDL) [16] and Universal Description Discovery and Integration (UDDI) [17]. Other Web service standards are built on these basic standards to provide higher-level functions and quality of service. Examples of these standards are WS-Security [18], WS-ReliableMessaging [19] and WS-AtomicTransaction [20].

The following sub-sections briefly describe the three basic Web Services standards.

4.2.1 SOAP

SOAP is an XML-based format for constructing messages in a transport independent way and a standard on how the message should be handled. SOAP messages consist of an envelope containing a header and a body. This format also defines a mechanism, known as SOAP faults, for indicating and communicating problems that occurred while processing the message.

The headers section of a SOAP message is extensible and can contain many different headers defined by different schemas. The extra headers can be used to modify the behavior of the middleware infrastructure. For example, the headers can include information about transactions that can be used to ensure that actions performed by the service consumer and service provider are coordinated.

The body section contains the content of the SOAP message. This data is specified in the WSDL describing the Web service. It is common to talk about SOAP in combination with the transport protocol used to communicate the SOAP message. For example, SOAP being transported using HTTP is referred to as SOAP over HTTP.

4.2.2 WSDL

WSDL is an XML-based interface definition language that separates function from implementation and enables design by contract as recommended by SOA. WSDL descriptions contain a port type (the functional and data description of the operations that are available in a Web service), a binding (providing instructions for interacting with the Web service through specific protocols, such as SOAP over HTTP), and a port (providing a specific address through which a Web service can be invoked using a specific protocol binding). It is common for these three aspects to be defined in three separate WSDL files, each importing the others.

The value of WSDL is that it enables development tools and middleware for any platform and language to understand service operations and invocation mechanisms. As with SOAP, the WSDL specification is extensible and provides for additional aspects of service interactions to be specified, such as security and transactions.

4.2.3 UDDI

UDDI is a standard for registering and searching web services within directories. First, a service provider uses UDDI to store the publisher, service description, location of the service, and the interfaces to access the service. Using these elements (as well as registry-specific service categorizations) consumers of web services can search for services. Then a consumer can use UDDI to locate an appropriate service. Finally, the consumer can connect to and use the service.

The UDDI standard defines interfaces to a directory that are themselves web services described by a WSDL. The original UDDI classification was based on a U.S. government taxonomy of businesses. However, recent versions of the UDDI specification have added support for custom taxonomies.

4.3 Implementing a SOA with Web Services

Web services are a technology that is well suited for implementing a SOA. As described in the previous section, Web services are self-describing and modular applications that expose business logic as services that can be published, discovered, and invoked over the Internet. Based on XML standards, Web services can be developed as loosely coupled application components using any programming language, any protocol, or any platform. This facilitates the delivery of business applications as a service accessible to anyone, anytime, at any location, and using any platform.

However, it is important to point out that Web services are not the only technology that can be used to implement a SOA. Additionally, Web services have been used to implement architectures that are not necessarily service-oriented.

5 QBox-Services Platform

As described in section 2, the QBox Foundation platform has some scalability problems: incompleteness of quality methods and lack of connectivity to integrate functionalities of existing tools. In this section, a new approach for the QBox platform is presented which, taking advantage of service-oriented mechanisms, addresses these problems.

5.1 General Description

The new Qbox-Services platform conserves all the functionalities of QBox-Foundation, except the measurement methods which have been externalized as services, as shown in Figure 9. This allows reusability of existing quality tools, making QBox-Services open to the integration of any user-defined or market-supplied tool.

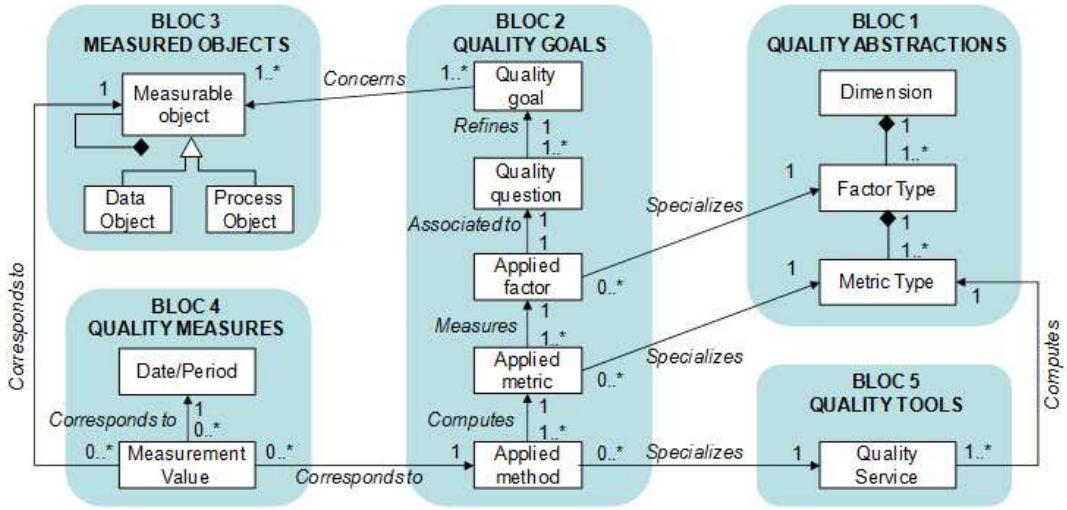

Figure 9 - Quality Assessment Meta-Model

The platform consists of three main components called Qbox-Foundation, QMediator and QManagement. These components, and other important concepts involved in the architecture, are sketched in Figure 10.

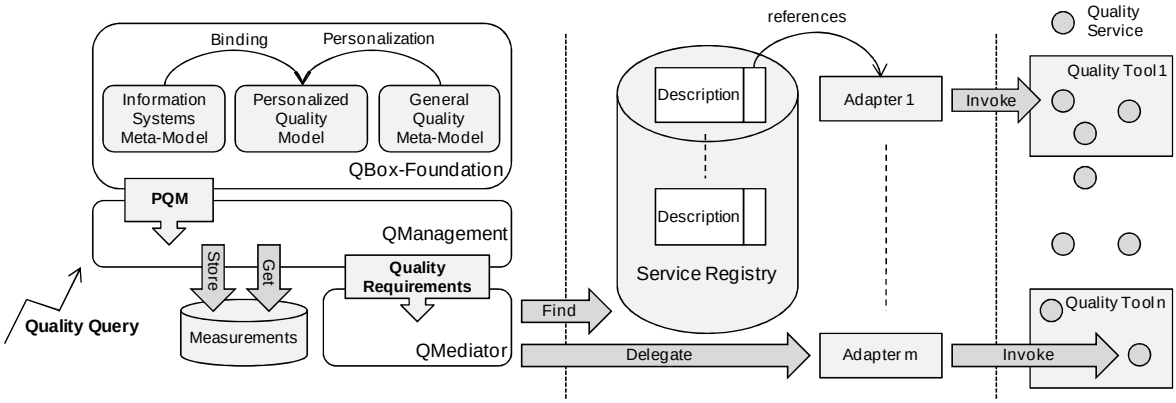

Figure 10 - QBox Service Architecture

The QBox-Foundation component inherits the functionalities to define quality goals and refine and specialize quality metrics according to these goals. Through these functionalities a quality expert can derive, from the generic quality meta-model, a Personalized Quality Model (PQM), i.e. a refined set of quality factors and metrics that correspond to specific quality goals and refer to specific IS objects.

As a result, a PQM embeds specific quality requirements that can be answered by executing appropriate quality services. Quality services may be of three types: measurement services that compute quality metrics; analysis services that analyze a set of measures and calculate complex indicators and improvement services that perform data updates in order to improve quality. Examples of those types of services are the calculation of the percentage of null values, the analysis of growing/decreasing tendencies in data freshness and the elimination of duplicate tuples, respectively.

The QMediator component provides functionalities for finding appropriate services for these requirements, enabling their execution and returning their results. To this end, it accesses a service registry that contains abstract service descriptions and access patterns of available quality services. A set of adapters implement these access patterns and invoke the quality services encapsulating technological details. QMediator acts as a mediator between quality requirements expressed in the PQM (e.g. the need to assess the metrics of a given goal) and the available quality services (especially those calculating such metrics).

The QManagement component executes required quality services for specific goals and provides an interface for analyzing results. Quality services may be periodically executed, may be punctually invoked by a user or may be triggered by another service (e.g. an analysis service that needs comparing some non-available measures). Result analysis is carried out by evaluating quality queries, called Qolap queries, defined over the PQM. QManagement includes a decision-support interface that allows browsing in the star-like quality model and posing quality queries. If some of the necessary measures are not available, the corresponding services may be executed in order to obtain them.

5.2 Mediation of Quality Services

QMediator acts as a mediator between quality requirements expressed in terms of the PQM and quality services listed in a service registry. The functionalities of quality tools are described as abstract quality services and a delegation mechanism binds an abstract quality service to a specific implementation in a given external quality tool.

A quality service as an implementation of a quality functionality that can be either custom implemented within an organization, or provided by external quality tools. An abstract quality service is the description of the quality functionality provided by a quality service. This description includes the quality concepts it addresses (dimensions, factors and metrics), the type of functionality it provides (measurement, analysis or improvement) and the supported IS object type (tables, XML files, etc).

An access pattern specifies the interface of an abstract quality service, that is, the way the service is invoked, its input and output parameters as well as its exception handling. An adapter implements an access pattern for a given quality service, providing the means for invoking a quality service and making transparent tool-specific technological details.

The service registry provides the mechanisms for the publication and discovery of abstract quality services according to specific quality requirements. For each abstract quality service, it also stores its access patterns and the way to invoke the adapters, providing access to the functionalities of quality tools.

The main responsibility of QMediator is, given some quality requirements, find and execute the quality services that best match these requirements. To this end, QMediator has to find in the registry the abstract quality services which declare fulfilling the given quality requirements, select the abstract quality services to use, get the corresponding access patterns from the service registry, delegate the invocation of the quality services to the adapters components, and finally consolidate the result of quality services invocations in a unified result. Figure 11shows a simplified sequence diagram of this process.

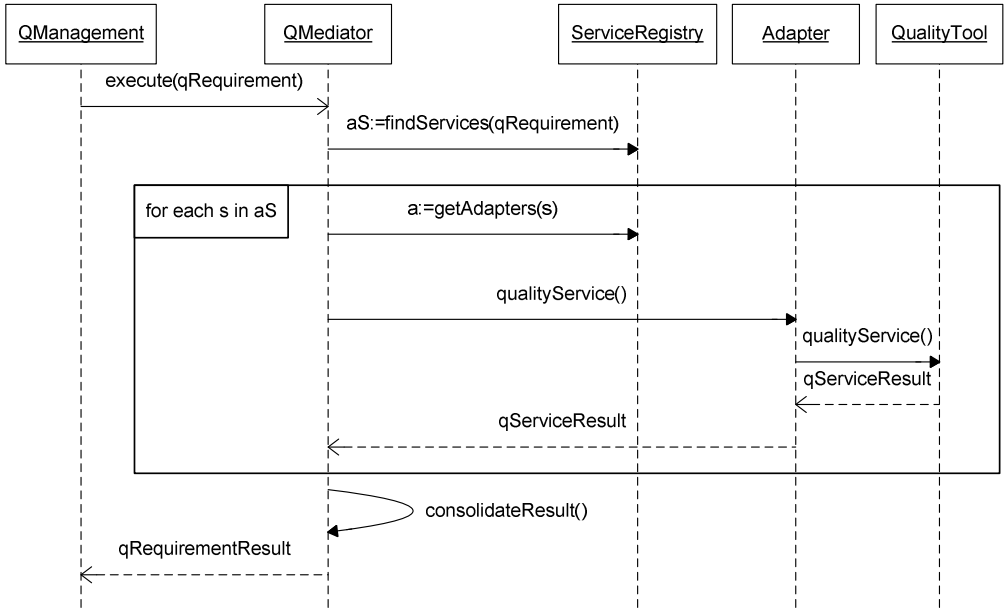

Figure 11 - Execution of a Quality Requirement

In this way, the library of measurement methods, that used to belong to the Qbox-Foundation component, is replaced by a collection of quality services that may be provided by external quality tools.

6 QBox-Services Prototype

This section describes the development of a prototype which implements the QBox-Services architecture. The prototype is composed of two main modules: QBox Services Core and Quality Services. Additionally, it relies on the QBox Foundation API, Apache juddi [22] and JBoss WS components [23]. Figure 12 shows a deployment diagram of the implemented prototype.

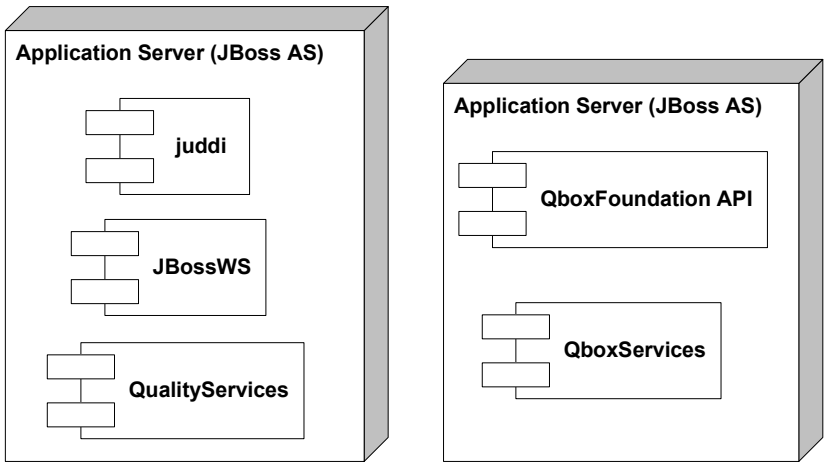

Figure 12 - QBox-Services Deployment Diagram

In the following sub-sections the prototype main modules are described.

6.1 Quality Services

The prototype includes quality services provided by the DataCleaner Project. Additionally, it integrates custom-developed quality services that, leveraging the DataCleaner quality services, implemented the calculus of some quality metrics.

6.1.1 DataCleaner Quality Services

As described in section 3.1, the DataCleaner Project provides methods for profiling and validating data taken from various types of data sources. In order to integrate the DataCleaner methods in the prototype, adapters were implemented in the Java programming language and exposed via the Web Service technology. As shown in Figure 13, given the modular design of the DataCleaner tool, the developed adapters rely on the logic implemented in the DataCleaner Core module, in the same way the DataCleaner Gui module does.

Figure 13 - Adapters accessing DataCleaner Logic

The adapters were implemented using the Java programming language, exposed via the Web Services technology using the JAX-WS [25] implementation provided by the JBoss Web Service (JBossWS) component, and deployed in a JBoss Application Server [24].

Although the DataCleaner projects supports a wide range of data sources, the methods included in the prototype works over relational databases accessed via JDBC Therefore, these methods need to receive, as input parameters, all the required data to allow DataCleaner to connect to databases through JDBC [26]. Table 7 presents and describes the JDBC required parameters.

Parameter Name	Description	Example
driver	Full qualified name of the jdbc driver	org.postgresql.Driver
connectionstring	Connection string to access the database	jdbc:postgresql://localhost:5432/myDB
schema	Schema	public
username	Username to access the database	postgres
password	Password associated to the username	postgres

Table 7 - JDBC Parameters

Table 8 presents the description and required parameters for one of the DataCleaner quality service, and Table 9 presents a summary of the DataCleaner methods included in the prototype.

Name	getEmptyValues	**Description**	It returns the number of empty values in the specified column.
Parameters			
Name	**Type**	**Direction**	**Description**
JDBC_PARAMETERS	N/A	IN	JDBC parameters as specified in Table 7.
tablename	String	IN	Table name
columnname	String	IN	Column name
emptyvalues	Long	OUT	Number of empty values in the specified column

Table 8 - getEmptyValues Quality Service

DataCleaner Categorization	Name
Standard Measures	getEmptyValues
	getHighestValue
	getLowestValue
	getNullValues
	getRowCount
Number Analysis	getGeometricMean
	getHighestValue
	getLowestValue
	getMean
	getStandardDeviation
	getSum
	getVariance
String Analysis	getCharCount
	getLowercaseChars
	getMaxChars
	getMaxWords
	getMinChars
	getMinWords
	getNonLetterChars
	getUppercaseChars
	getWordCount
Validation	getDictionaryValidation
	getJavascriptValidation
	getNotNullValidation
	getRangeValidation
	getRegularExpressionValidation

Table 9 - Summary of DataCleaner Methods included in the prototype

6.1.2 Custom Quality Services

In order to include in the prototype quality services which implemented some specific metrics, a number of custom quality services were developed. The custom quality services were implemented using the Java programming language and leveraging the functionality provided by the Datacleaner quality services. They were also exposed via the Web Services technology using the JAX-WS implementation provided by the JBossWS component, and deployed in a JBoss Application Server.

As an example, Table 10 presents the description and required parameters for one of the custom quality services.

Name	getEmptyValues	**Description**	It returns a value between 0 and 1, which represent the percentage of syntactically correct values in the column according to the specified dictionary.
Parameters			
Name	**Type**	**Direction**	**Description**
JDBC_PARAMETERS	N/A	IN	JDBC parameters as specified in Table 1.
Tablename	String	IN	Table name
Columnname	String	IN	Column name
Dictionarypath	String	IN	Dictionary path
syntacticcorrectnessratio	Double	OUT	Value between 0 and 1, representing the percentage of syntactically correct values in the column according to the specified dictionary.

Table 10 - getSyntacticCorrectnessRatioDictionary Quality Service

Table 11 presents a summary of the Custom Quality Services included in the prototype and the metric they implement.

Metric	Name
SyntacticCorrectnessRatio	getSyntacticCorrectnessRatioDictionary
	getSyntacticCorrectnessRatioRegexp
DensityRatio	getDensityRatio
RelationIntegrityRatio	getRelationIntegrityRatio

Table 11 - Summary of Custom Quality Services

6.2 Services Registry

The prototype leverages the Apache juddi project as the implementation of a UDDI registry. In order to run the prototype some data have to be included in the UDDI registry.

First, as shown in Table 12, two service providers were published in the UDDI registry: Datacleaner and QBoxFoundation.

Name	Description
DataCleaner	DataCleaner is an Open Source application for profiling, validating and comparing data.
QBoxFoundation	QboxFoundation is a metadata platform for quality assessment based on the Goal-Question-Metric (GQM) paradigm.

Table 12 - Registered Services Providers

Additionally, in order to classify and describe the published services in the registry, according to the functionality they provide, three taxonomies were published. Table 13 presents and describes these taxonomies.

Name	Description
qbox-org:quality:indicator	Taxonomy to represent quality indicators.
qbox-org:quality:operation	Taxonomy to represent quality operations.
qbox-org:object:type	Taxonomy to represent measurable object types.

Table 13 - Registered Taxonomies for Categorizing Quality Services

Finally, some of the quality services described in previous sections were published in the registry. Table 14 presents these quality services along with its classification according to the taxonomies that were published in the registry.

Provider	Name	Indicator	Operation	Object Type
QBox Foundation	SyntacticCorrectnessRatioDictionary	SyntacticCorrectnessRatio	calculus	column
	SyntacticCorrectnessRatioRegexp	SyntacticCorrectnessRatio	calculus	column
	DensityRatio	DensityRatio	calculus	column
	RelationIntegrityRatio	DensityRatio	calculus	column
DataCleaner	NullValues		calculus	column
	RowCount		calculus	table
	DictionaryValidation		calculus	column
	JavascriptValidation		calculus	column
	RegularExpressionValidation		calculus	column

Table 14 - Registered Quality Services and its Categorization

6.3 QBox Services Core

The main component developed for the QBox-Services prototype is the QBox-Service component which in turn is split in a set of more specific components. These sub-components are presented in Figure 14 along with the dependency relationships among them.

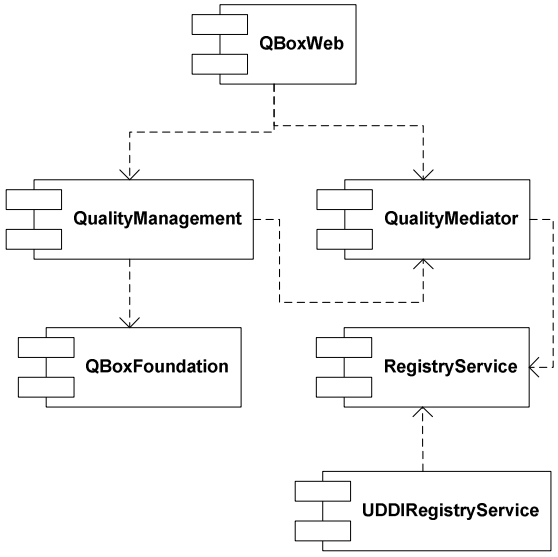

Figure 14 - QBox Services Components

The **Registry Service** component encapsulates the access to a services registry; specifically the UDDI Service Registry component implements the access to the juddi service registry.

The **Quality Mediator** component provides an interface for finding appropriate services for specific quality requirements. This component relies on the Registry Service component to find the quality services which fulfill these requirements, and the access pattern and adapters to access these services. The Quality Mediator component delegates to the adapters the invocation of the quality services.

The **Quality Management** component receives quality requirements in terms of the PQM and, relying on the QBox Foundation API and the Quality Mediator components, transforms these personalized requirements in specific quality requirements.

Finally, the **QBox Web** component is a Java Web Application which, relying in the Quality Mediator component and in the Quality Manager component, allows the search and invocation of quality services registered in the service registry.

Figure 15 presents the QBox Web user interface which provides functionalities for searching, viewing the details and invoking services in the registry according to specific calculus requirements.

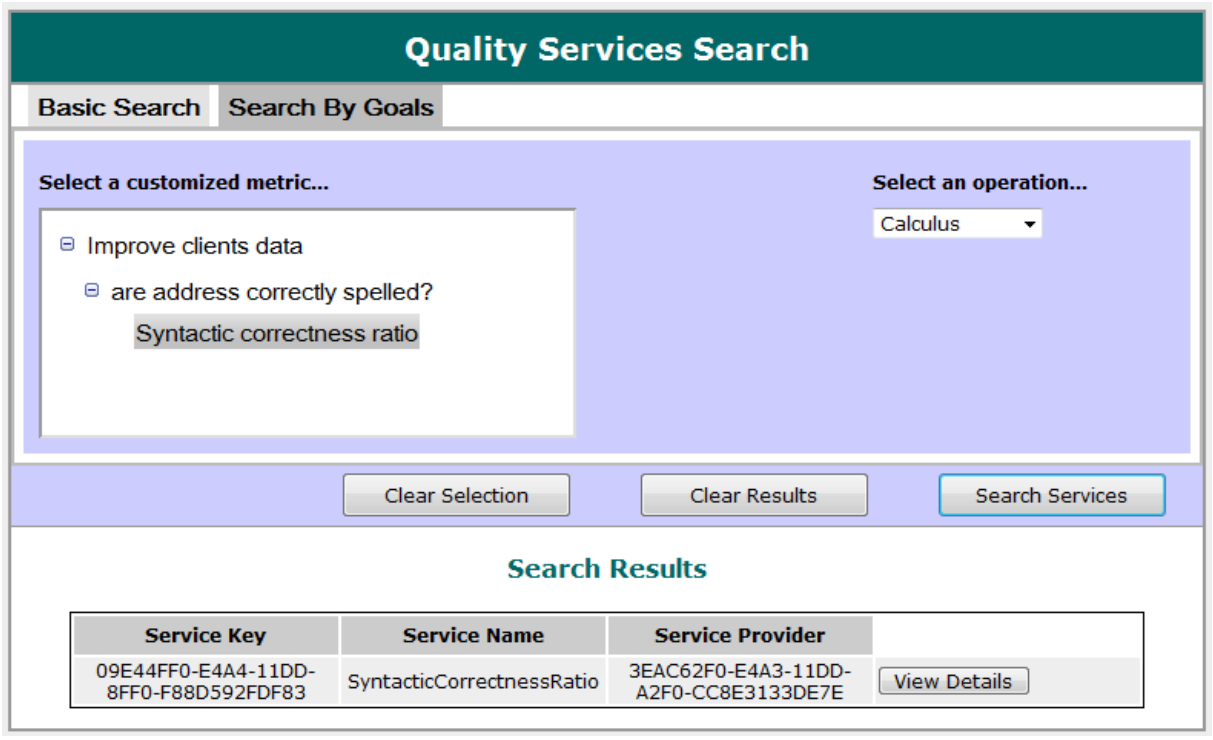

Figure 15 - QBox Web User Interface

7 Conclusions and Future Work

Although QBox-Foundation constitutes a relevant and useful platform to deal with data quality in information systems, some experiments revealed limitations which lead to scalability problems. In this report, a new approach for the QBox platform is presented which, taking advantage of service oriented mechanisms, addresses the scalability problems and allows the reuse of existing quality tools. This new approach also allows interoperation between user-defined quality methods and external quality tools, giving the user the possibility to access a larger set of quality functionalities and reducing implementation efforts.

The new platform conserves the facilities for defining and personalizing a quality model according to user quality goals and the underlying IS. The Qbox-Foundation component was modified for decoupling methods management functionalities, directing its focus to the management of the personalized quality model. A new QMediator component was developed which, using Web Service technologies, manages a dynamic library of quality tools (catalogued in a service registry) and provides functionalities for finding quality services according to quality requirements. Adapters for several tools were also developed and registered in the service registry. Finally, a QManagement component was also developed which receives quality requirements in terms of the PQM and, relying on the QBox Foundation and the Quality Mediator components, transforms these personalized requirements in specific quality requirements.

Further work can focus on the formalism for expressing quality queries and experimentation in various application domains.

8 References

[1] Basili, V., G. Caldiera and H.D. Rombach (1994). The Goal Question Metric Approach. Encyclopedia of Software Engineering, 528-532, John Wiley & Sons, Inc.

[2] Vassiliadis, P., M. Bouzeghoub and C. Quix (2000): Towards Quality-oriented Data Warehouse Usage and Evolution. *Information Systems*, 25(2): 89-115.

[3] Akoka, J., L. Berti-Equille, O. Boucelma, M. Bouzeghoub, I. Comyn-Wattiau, M. Cosquer, V. Goasdoué-Thion, Z. Kedad, S. Nugier, V. Peralta and S. Sisaid-Cherfi (2007). A Framework for Quality Evaluation in Data Integration Systems. *9th International Conference on Enterprise Information Systems (ICEIS'2007)*, Funchal, Portugal.

[4] Etcheverry, L., V. Peralta, V. and M. Bouzeghoub (2008). Qbox-Foundation: a Metadata Platform for Quality Measurement. *4ème Atelier Qualité des données et des Connaisances (QDC'2008)*, Nice, France.

[5] DataCleaner Project
http://datacleaner.eobjects.org/ – March 2009

[6] Aggregate Profiler
http://sourceforge.net/projects/dataquality/ – March 2009

[7] Open Data Quality
https://open-dm-dq.dev.java.net/ – March 2009

[8] Mural
https://mural.dev.java.net/ – March 2009

[9] Data Integrator
https://open-dm-di.dev.java.net/ – March 2009

[10] Talend Open Profiler
http://www.talend.com/products-data-quality/talend-open-profiler.php – March 2009

[11] Power MatchMaker
http://www.sqlpower.ca/page/MatchMaker – March 2009

[12] James MCGovern, Oliver Sims, Ashish Jain, Mark Little. Enterprise Service Oriented Architectures: Concepts, Challenges, Recomendations. Springer. 2006. ISBN: 978-1-4020-3704-7

[13] Martin Keen, Jonathan Bond, Jerry M Denman, Stuart Foster, Stepan Husek, Ben Thompson, Helen Wylie. Patterns: Integrating Enterprise Service Buses in a Service-Oriented Architecture. IBM RedBooks 2005. ISBN 0738492930. http://www.redbooks.ibm.com/abstracts/sg246773.html – March 2009

[14] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, POEI Krogdahi, Dr Min Luo, Tony Newling. Patterns: Service-Oriented Architecture and Web Services. IBM RedBooks 2004. ISBN 073845317X. http://www.redbooks.ibm.com/abstracts/sg246303.html – March 2009

[15] SOAP – Simple Object Access Protocol
http://www.w3.org/TR/soap/ – March 2009

[16] Web Service Definition Language (WSDL)
http://www.w3.org/TR/wsdl – March 2009

[17] Universal Description Discovery and Integration (UDDI)
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec – March 2009

[18] Web Services Security
http://www.oasis-open.org/committees/wss/ – March 2009

[19] Web Services Reliable Messaging
http://docs.oasis-open.org/ws-rx/wsrm/v1.2/wsrm.html – March 2009

[20] Web Services Atomic Transaction
http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec/wstx-wsat-1.1-spec.html – March 2009

[21] Web Services Architecture Working Group
http://www.w3.org/2002/ws/arch/ – March 2009

[22] Apache juddi
http://ws.apache.org/juddi/ – March 2009

[23] JBoss Web Services
http://www.jboss.org/jbossws – March 2009

[24] JBoss Application Server
http://www.jboss.org/jbossas – March 2009

[25] JavaTM API for XML-Based Web Services
http://jcp.org/en/jsr/detail?id=224 – March 2009

[26] JDBC Overview
http://java.sun.com/products/jdbc/overview.html – March 2009