Masterarbeit

Data Access and Integration Services for Cloud Computing Platforms

verfasst von

YURIY KANIOVSKYI, BAKK. TECHN.

angestrebter akademischer Grad

Diplom-Ingenieur (Dipl.-Ing.)

Matrikelnummer	0506623
Studienkennzahl lt. Studienblatt	A 066 940
Studienrichtung lt. Studienblatt	Scientific Computing
Betreuer	Univ.-Prof. Dipl.-Ing. Dr. Siegfried Benkner

Wien, Februar 2013

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstndig und nur unter Benutzung der angegebenen Literatur und Hilfsmittel angefertigt habe. Wörtlich übernommene Stze und Satzteile sind als Zitate belegt, andere Anlehnungen hinsichtlich Aussage und Umfang unter Quellenangabe kenntlich gemacht. Die Arbeit hat in gleicher oder hnlicher Form noch keiner Prüfungsbehörde vorgelegen und ist auch noch nicht veröffentlicht.

Statutory declaration

I herewith declare that I have completed the present thesis independently making use only of the specified literature and aids. Sentences or parts of sentences quoted literally are marked as quotations; identification of other references with regard to the statement and scope of the work is quoted. The thesis in this form or in any other form has not been submitted to an examination body and has not been published.

Vienna, 2013

Yuriy Kaniovskyi

Acknowledgment

I would like to thank all those who made this thesis possible.

Foremost, I would like to express my sincere gratitude to my adviser Prof. Siegfried Benkner for his support of my thesis and research, for his guidance, patience, motivation, enthusiasm and immense knowledge. I thank him for inviting me to participate in the VPH-Share project, expanding my knowledge tenfold.

I would also like to express my deepest gratitude to my colleague and friend Dr. Martin Köhler at the University of Vienna, for his encouragement, insightful comments and undying support. I thank him in particular for his assistance in writing this thesis.

I thank my colleagues at the Research Group Scienfitic Computing at the University of Vienna for numerous discussions that have improved my work. Special thanks to Martin Paul for the technical support.

I am indebted to my family for their unconditional love, encouragement and patience. I am grateful to my father, Yuriy Kaniovskyi Sr., my mother, Irina Kaniovskaya and especially my older brother Serguei, whose support of this thesis has been immense.

Last but not the least, I would like to thank my dear friends for their strong moral support and generally making my life a happy experience: Dr. Alexander Wöhrer, Lukas Spitaler (a.k.a. Bamboo), Fabian Störk, Caro Apfelbach, Felix Tutzer, Philip Madeiski, Wolfgang Thaler, Christoph Giacomuzzi, Kinga Wozniak, Maddalena Rifesser, Felix Federer, Tanja Stoßfellner, Ivonne Planker, Alexander Mich, and the DJs Alexander Hell and Harald Cronst for ambient support – thank you!

Abstract

Data integration plays an important role in today's scientific community. To extract knowledge relevant to their domain, researchers often seek collaboration to tap various data sources throughout the globe. The European VPH-Share project seeks to combine geographically scattered clinical data sources through scientific workflows. The VPH-Share project develops data access and integration services for exposing clinical data sources. The services consist of various technologies, in particular the OGSA-DAI middleware, which provides a web service interface for accessing and integrating data sources. The SOAP-standard based interface of OGSA-DAI, however, is outdated, and lacks support for the current web service standards. This thesis describes the data access and integration services architecture that meets the project requirements. These make it a necessity to implement a new state-of-art SOAP-based interface.

This work presents design and implementation of the new interface based on the Apache CXF web service framework. The developed services are part of the Vienna Cloud Environment (VCE) software release. The VCE integrates the service implementation and allows utilization of different communication patterns (SOAP/REST), as well as supports provisioning of the services as virtual appliances. In addition to the comparison between the old and the new implementations, the experimental evaluation provides a study of service performance in different cloud environments.

Kurzfassung

Datenintegration spielt eine wichtige Rolle in der heutigen wissenschaftlichen Gesellschaft. Um relevantes Wissen zu ihrer Domäne zu extrahieren, suchen Forscher oft die Zusammenarbeit, damit verschiedene Datenquellen auf der ganzen Welt erschlossen werden können. Das Europäische VPH-Share Projekt soll geografisch verstreute klinische Datenquellen über wissenschaftliche Arbeitsabläufe kombinieren. Das Projekt besteht unter anderem aus Datenzugriffs- und Integrations-Services um klinische Datenquellen ins Netz auszusetzen. Diese Dienste bestehen aus verschiedenen Technologien, insbesondere der OGSA-DAI-Middleware, die eine Web-Service-Schnittstelle für den Zugriff und die Integration von Daten-Quellen zur Verfügung stellt. Die auf SOAP-Standard basierte Schnittstelle des OGSA-DAI ist jedoch überholt, und es fehlt die Unterstützung für die aktuellen Web-Service-Normenwerke. Diese Arbeit beschreibt die Datenzugriff- und die Integration-Services-Architektur, die die Anforderungen des Projekts erfüllt. Diese machen es jedoch notwendig, die auf SOAP-basierte Schnittstelle auf den neuesten Stand der Technik zu implementieren.

Diese Arbeit präsentiert das Design und die Umsetzung der neuen Schnittstelle basierend auf dem Apache CXF Web Service Framework. Die entwickelten Dienste sind Teil des Vienna Cloud Environment (VCE) Software-Releases. Das VCE integriert die Service-Implementierung und ermöglicht die Nutzung unterschiedlicher Kommunikationsmodelle (SOAP/REST), weiters unterstützt VCE die Bereitstellung der Dienste als 'Virtual Appliances'. Neben dem Vergleich zwischen den alten und den neuen Implementierungen bietet die experimentelle Evaluierung eine Studie der Leistungserbringung in verschiedenen Cloud-Umgebungen.

Curriculum Vitae

Personal Information

Name	Yuriy Kaniovskyi
Email	Yuriy.Kaniovskyi@univie.ac.at
Languages	Russian, German, English, Italian, Ukrainian

Work experience

Occupation	Project Participant
Dates	August 2011 until now
Responsibilities	Implementation of the WP3 data web services for the european VPH-Share project
Employer	Research Group Scientific Computing, Faculty of Computer Science, University Of Vienna, Währinger Straße 29, 1090 Vienna, Austria
Type of sector	Research
Occupation	Tutor
Dates	Winter semester 2011 and 2012
Subjects	Grid & Cloud Computing (2011 and 2012) and Software Engineering (2012)
Employer	University Of Vienna, Währinger Straße 29, 1090 Vienna, Austria
Type of sector	Teaching
Occupation	Research Assistant
Dates	September 2009 until July 2011
Responsibilities	Reseach and development of data access and integration services and MapReduce applications
Employer	Research Group Scientific Computing, Faculty of Computer Science, University Of Vienna, Nordbergstrasse 15/C/3,1090 Vienna, Austria
Type of sector	Research
Occupation	Internship
Dates	Summer 2006
Responsibilities	Design and Implementation of web pages and applications in HTML, PHP and SQL
Employer	Akanai OHG d. Tumler Alexander & Co. , Bahnhofwiesen 6, I-39014 Burgstall BZ , Italy

Type of sector	Commercial
Occupation **Dates** **Responsibilities** **Employer** **Type of sector**	Internship Summer 2004 Design and Implementation of Webpages in HTML,PHP and SQL The Lounge interactive design, Hofmühlgasse 17/1/3, A-1060 Vienna, Austria Commercial

Education

Organisation **Dates** **Main topics covered** **Location**	Genias Benelux bv and Vrije Universiteit Amsterdam July 2012 Contrail summer school on Cloud Computing Systems and Applications Almere, the Netherlands
Organisation **Dates** **Title awarded** **Specialization** **Fields of attention**	University of Vienna March 2008 until now Master of Science in Information Technologies Scientific Computing Grid & Cloud Computing, Software Engineering, Machine Learning, High Performance Computing
Organisation **Dates** **Title awarded** **Specialization** **Fields of attention**	University of Vienna October 2005 until January 2008 Bachelor of Science in Information Technologies Software and Information Engineering Software Engineering, Design patterns, Project management, Object Oriented Programming
Organisation **Dates** **Title awarded** **Fields of attention**	Gewerbe Oberschule (technical high school) 'Max Valier', Bolzano, Italy 2000 - 2005 Industrial computer expert Informatics, statistics, mathematics, electronics

List of publications

- Martin Köhler, Richard Knight, Siegfried Benkner, Yuriy Kaniovskyi and Steven Wood.
  The VPH-Share Data Management Platform: Enabling Collaborative Data Management for the Virtual Physiological Human Community.
  in *The 8th International Conference on Semantics, Knowledge & Grids*, *IEEE*, USA.
  October 2012.

- Siegfried Benkner, Jesus Bisbal, Gerhard Engelbrecht, Rod D. Hose, Yuriy Kaniovskyi and Martin Köhler, Carlos Pedrinaci and Steven Wood.
  Towards collaborative data management in the VPH-Share project.
  in *Proceedings of the Intl. Workshop on Cloud Computing Projects and Initiatives, in conjunction with Euro-Par*, *Springer*, Bordeaux, France.
  August 2011.

- Martin Köhler, Yuriy Kaniovskyi and Siegfried Benkner.
  An adaptive framework for the execution of data-intensive MapReduce applications in the Cloud.
  in *The First International Workshop on Data Intensive Computing in the Clouds (DataCloud 2011)*, *IEEE*, Anchorage, Alaska, USA.
  May 2011.

- Martin Köhler, Yuriy Kaniovskyi, Siegfried Benkner, Volker Egelhofer and Wolfram Weckwerth.
  A cloud framework for high troughput biological data processing.
  International Symposium on Grids and Clouds, PoS(ISGC 2011 & OGF 31)069, Taipei, Taiwan.
  March 2011.

List of projects

- **Name**: Virtual Physiological Human: Sharing for Healthcare - A Research Environment.
  Duration: 01.03.2011 28.02.2015
  Reference Number: 269978
  Description: Collaborative project partially funded within the Seventh Framework Programme by the European Commission. VPH-Share is developing an organisational framework for the widespread integration of VPH services, with the goals to expose and share clinical data and knowledge, jointly develop multiscale models for the composition of new VPH workflows and facilitate collaborations within the VPH community.

Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

I have an almost religious zeal...
not for technology per se, but for the Internet which is for me,
the nervous system of mother Earth,
which I see as a living creature, linking up.
–Dan Millman

1.1 Motivation

In today's society, knowledge is highly decentralized. Many research institutions autonomously pursue similar or related research questions. Many practitioners such as enterprises, engineers, physicians and other specialists work on similar problems on a daily basis. It is therefore not surprising that these localized activities often resort to different tools and solutions. Even within the same organization it is often the case that many different standards and technologies are used simultaneously. The fact that these data are often encoded in different media and storage formats makes processing such data a challenging task. One of the persistent challenges facing the modern society is therefore how to use knowledge scattered among many heterogeneous and decentralized sources effectively and efficiently.

Using data in a highly decentralized way confers many advantages due to specialization. One advantage is flexibility and the ability to be easily tailored to local needs. Another advantage is that processing smaller databases requires scaling computing resources out instead of scaling up, which is easily affordable to small organizations. Heterogeneous and decentralized systems, however, have one major shortcoming, which is the absence of a single data access point that can be queried and analyzed. Another shortcoming of heterogeneous and decentralized systems is a lack of global perspective on similar or related issues. Furthermore, dissem-

inating new knowledge to widely scattered and separately managed sources poses a challenge of its own.

Following the recent years, the research community shifted its focus from the high performance grid to the data intensive cloud. Popular fields of application include bioinformatics, astronomy, seismology and physics. Researchers in these fields have to deal with a key bottleneck: the interface with a distributed and collectively large amount of data. As in many other scientific collaborations, data integration is a key issue, since modern applications of individual sources require the integrated data of other, private or otherwise, sources in order to benefit from additional input.

Two approaches to tackling the challenge of data integration by using scattered and heterogeneous sources involve data warehousing and virtual data integration. Both concepts work with a unified mediated scheme applied to scattered but related data sources. Both concepts involve a series of mappings identifying and combining separated data sources in one integrated domain. Despite many similarities between them and the existence of numerous intermediate system types, there are cases in which one concept is clearly superior to the other. The context of this work, described in detail in Section 2.1, requires a solution based on virtual data integration. Key differences between data warehousing and virtual data integration will be discussed in Section 2.3.

New challenges arise with respect to integrating differently located data sources into the cloud and to providing seemingly transparent access to a virtually combined data source (Section 3.1). The importance of relational database management systems (RDBMS) cannot be ignored here. OGSA-DAI [1], a data integration middleware, plays a key role in exposing heterogenous data sources, including RDBMS, into the cloud. This raises the question for the research community as well as for business enterprises of how RDBMSs and their built-in technologies can be adapted to fit to this environment and, more importantly, provide a unified interface for their use.

1.2 Aim of the thesis

In this thesis, a virtual data integration approach based on OGSA-DAI is applied to expose various scattered data sources into a cloud environment. Cloud applications, such as OGSA-DAI, typically provide its resources as a service. The widely spread service-oriented Architecture (SoA)[2] design is a coarse-grained system architecture, in which loosely coupled, stateless and autonomous services communicate via structured web messages. This architecture provides the middleware a high-level interface, hiding the underlying logic, but guaranteeing its interoperability through the Internet. Due to the requirements of the project context described in the next chapters, several changes to the middleware are necessary. These apply mainly in the area of the service interface of OGSA-DAI.

The specific application scenario involves the VPH-Share project [3] that deals with collecting, structuring and disseminating data from biomedical research. The developed baseline solution, however, can be used in other fields, in which large amount of data distributed among several sources needs to be pooled and analyzed. Several example of potential application are briefly discussed. The presented services will adhere to the Vienna Cloud Environment (VCE) [4] that, in addition to holding the data service implementation, will provide the tools necessary to deploy and configure the services into a runtime environment. The VCE, consisting of several layers, should then be provided as a virtual appliance, i.e. a preconfigured virtual image, which can be hosted on a cloud infrastructure.

This thesis presents the architecture and composition of the overall data infrastructure and describes in detail the design and implementation process of the modifications to the service interface of OGSA-DAI, including its integration into the VCE. The work resolves around extending, adjusting, configuring, deploying and implementing this new service layer to meet the context requirements. As this implementation uses a different service interface then OGSA-DAI's default, the development team of OGSA-DAI may see this solution to be fit for a contribution to their software.

In the following, data access and integration services will be referred to as data services.

1.3 Used technologies

Following is a table of technologies used for this thesis:

Table 1.1: Technologies used

Coding	Java, JUnit 4, Eclipse IDE
Hosting OS	Ubuntu 11.04, CentOS 6.3
Runtime environment	JDK/JRE 7u11, Apache Tomcat 7.0.25
DBMS	MySQL 5.5 community edition
Middleware	OGSA-DAI 4.1 (including OGSA-DQP, OGSA-Views and OGSA-RDF)
Interface	Apache CXF 2.4.1 (including Spring Framework 3.0.1)
Support tools	Apache Ant 1.8.3, log4j 1.2.8
Cloud	Atmosphere IaaS [5], Amazon EC2

1.4 Outline and structure

Following this introduction, the remainder of this thesis is structured as follows:

- Chapter 2 describes the research context of the work. This includes insights into topics such as data integration, cloud computing and software oriented architecture that are relevant for the later chapters.
- Chapter 3 introduces the architecture of the data services. After discussing the related challenges, an abstract architecture of the system is presented. It contains several coarse-grained layers that are further analyzed, resulting in a final system architecture.
- Chapter 4 presents the requirements, the scope, the constraints and the boundaries of the assignment. The description of the requirements for the implementation is followed by use case analysis with identification of the primary use cases and description of testing scenarios.
- Chapter 5 describes the design of the software in terms of further architecture analysis and UML modeling. To elaborate the use cases of the previous chapter, sequence diagrams are provided, following an implementation modeling via a class diagram.
- Chapter 6 presents the implementation of the described system. The chapter describes technical details across the defined system components, explains how they work together and gives an insight on issues that arose during the implementations.
- Chapter 7 provides and experimental evaluation of the system. Its includes performance tests with respect to the different web service framework as well as the testing of the use case scenarios from Chapter 2.

Chapter 2

Research context

Just as we could have rode into the sunset,
along came the Internet, and it tripled the significance of the PC.
–Andy Grove

This chapter provides the background on topics that are relevant for this work. First, the context of the work, a European research project, is presented. A brief insight is provided into the data integration theory, where a discussion of the two main approaches are described. The next section will introduce the reader to the systems environment - the cloud, while the last section describes the service oriented approach as an architectural software model for the cloud.

2.1 The VPH-Share project

> The Virtual Physiological Human (VPH) is a methodological and technological framework that, once established, will enable collaborative investigation of the human body as a single complex system. The collective framework will make it possible to share resources and observations formed by institutions and organizations creating disparate, but integrated structural and functional models of the living human body. This system will enable academic, clinical and industrial researchers to improve their understanding of human physiology and pathology, to derive predictive hypotheses and simulations, develop and test new therapies, with the eventual outcome of better disease diagnosis, treatment and prevention tools in healthcare. [3]

Diagnosis and treatment of complex or rare diseases requires specific expertise that is rarely found in one place. Typically, the required knowledge and experience is scattered among many

researchers and practitioners working in different institutions and countries. Combining this knowledge requires an infrastructure for collecting, structuring and disseminating the relevant information, with the ultimate aim of improving the chances of an accurate diagnosis and successful treatment.

The ongoing VPH-Share project, funded by the Seventh Framework Programme of the European Commission, is a clinical data infostructure consisting of specialized web services for finding, extracting and processing clinical data. The main idea is to construct a patient avatar to a medical researcher, for a particular condition, through comprehensive filtering and processing of live clinical data. This vision is depicted in Figure 2.1.

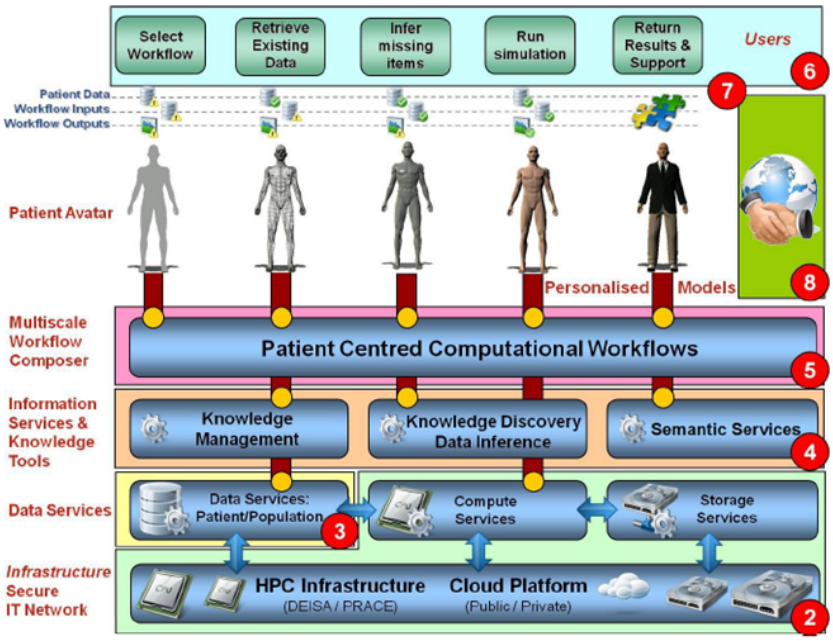


Figure 2.1: The VPH-Share project consists of a compositional service platform. Services are devided into several types ranging from data to computational workflow services. The underlyring system infrastructure is a hybrid cloud composed of private cloud providers and Amazon EC2 Infrastructure as a Service. The work of this thesis is related to the data service layer (3) of VPH-Share. [3]

The data involved typically are multi-scale in a qualitative as well as a quantitative sense, and multi-modal. The scope of coverage ranges from micro to macro biology, comprising of patient data, medical images and biomedical signals. The primary aim of the project is to provide accessible and comprehensive tools for data handling, reduction and representation. These basic functionalities are supplemented with tools for statistical inference and mathematical modeling, thus facilitating the process of scientific discovery and systematization.

The aim of VPH-Share is to provide an adequate IT infrastructure for the management of all processes linked to research, diagnosis and treatment of diseases. The mission statement of the VPH-Share project is stated as follows:

- expose and share biomedical data and knowledge;
- develop multi-scale models for the composition of new workflows;
- facilitate collaborations within the VPH community.

One of the main challenges of the project is to provide comprehensive analytical tools for researchers, while safeguarding personal clinical data protected by law. This makes the classic data warehouse solution inadequate, as live data cannot be moved from the location it was generated in, typically a clinic. Instead, a cloud solution, in which the data is made available through a virtual database to a scientific workflow, is envisaged. At the heart of this IT infrastructure is a cloud that provides a set of application platforms and specialized software solutions for collecting and processing data, in which sensitive data is not disseminated, while the relevant scientific and clinical discoveries are.

The services described here are related to the work package 3 of the VPH-Share project as part of the data infrastructure. They will be provided within an integrated framework, described in the next section.

2.2 The Vienna Cloud Environment

The Vienna Cloud Environment (VCE) has been developed at the University of Vienna to provide an integrated framework for provisioning virtual data service appliances and other types of virtual appliances. VCE also provides a high-level application programming interface with Java, C# and C bindings that may be used to construct advanced applications from application, data, workflow, and MapReduce services. In words of the lead developer [4]:

> VCE is a prototype framework for exposing virtual appliances as Web services. The VCE follows the Software as a Service (SaaS) model and relies on the concept of virtual appliances to provide a common set of generic interfaces to the user while hiding the details of the underlying software and hardware infrastructure. The service provisioning framework in VCE has been developed on top of the Vienna Grid Environment (VGE), a service oriented infrastructure for virtualizing scientific applications and data sources as Web services. VGE was developed in context of the European projects GEMSS and @neurIST and has been cloud-enabled by supporting virtual appliances. Currently VCE comprises a set of preconfigured virtual appliances, including an installation of the VCE Web Service Environment and a Client Environment.

This thesis describes the virtual data services that contribute to the VCE. The services are designed to be utilized by higher-level services that require the data. They can be used in

conjunction with other VCE appliances, such as scientific workflow appliances based on the WEEP workflow engine [6] and HPC application appliances, see Figure 2.2.

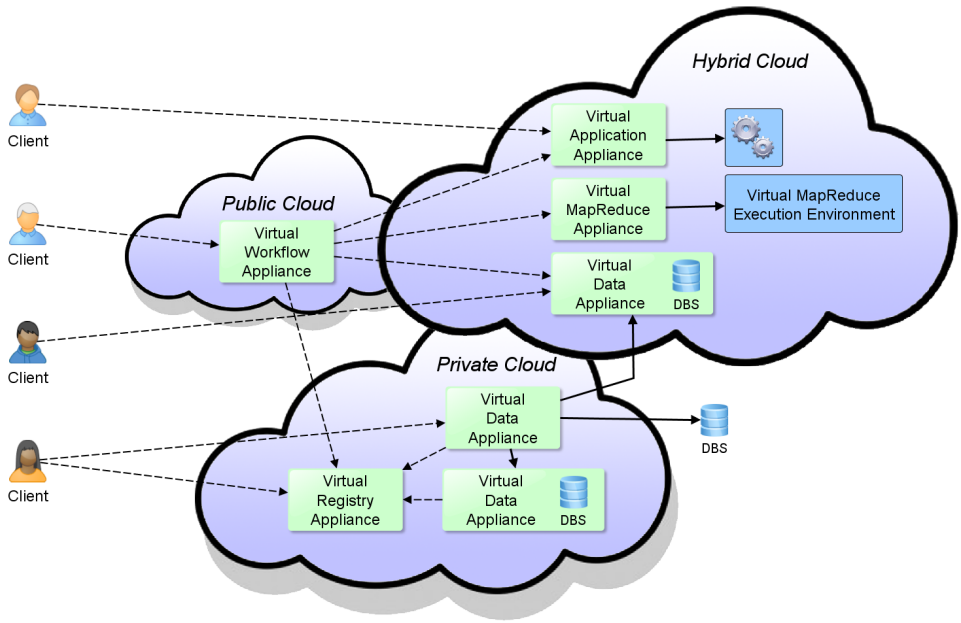


Figure 2.2: The VCE is based on a virtual appliance model within a component-based service provision framework, which supports the configuration of application, data, and workflow services from a set of basic service components providing capabilities, such as job or query execution, data transfers, QoS negotiation, data staging, and error recovery. [4]

Another key feature of the VCE is enabling service consumption through the two most popular communication models for services - SOAP messaging, described below, and REST access model [7]. The data browser can be used to navigate through the data set and view its schema. The deployment and configuration utility gives users the ability of easily setting up the data services through a graphical and command-line user interface. Other components of the VCE provide a rich logging, debugging and development environment.

The VCE data services will also support a semantic mechanisms for data exploration and extraction. In this relation a querying interface is to be provided and methods of querying a SPARQL endpoint implemented. The goal is to provide a unified access point for both relational and the related semantic data. Similar to relational federation, a SPARQL-DQP engine is planned for combination of several endpoints into a virtual one. The details of this requirement are provided in Section 3.4.1. The following section describes challenges with regard to the proposed system.

2.3 Data access and integration

Today, computing faces the challenge of synthesizing data scattered and heterogeneous sources. The the aim is to provide a unified, transparent and flexible view on the data. The demand for such a view originate both in scientific and the business communities, with biomedical and clinical research community being no exception (see, for example, [8]). The theory of Data Integration, as part of database theory, lays a theoretical groundwork for managing data from autonomous, heterogeneous, geographically distributed, and constantly evolving sources [9]. Recent developments in this theory proposed new approaches for data decoupling, for example, a unified query-interface by means of a middleware that resolves semantic conflicts between heterogeneous data sources and promotes the development of integrated data models.

Sheth and Larson [10] introduced techniques to provide a Federated Database Management System (FDBMS) that meets the challenges related to unifying such data sources. A federation topology has been chosen because it allows each data providing institution to retain control over their data source. Data integration systems utilize two main techniques. The goal of both is to provide an interface through data abstraction and allow users to query multiple noncontiguous, heterogeneous data sources transparently.

A popular solution is a data warehouse that offers extract, transform, load (ETL) operations on heterogeneous sources and converts them it into a single source with a common schema. Several issues arise with delegation of data to a warehouse: it needs to be kept up to date, large data sizes may constrain efficient processing, and, finally, data conversions may alter the data schema. The ETL loads through query interfaces of the converted data and not the original source.

In the recent years, the development of a more loosely coupled architecture for federated data management was favored. The idea is to consolidate several sources in a real-time manner using a mediation schema. The schema maps data columns from different data sources to the global virtual table it represents. The data is then pulled directly from the original source, without having to load, store and alter it, as in the lengthy ETL processes. This approach requires a middleware that implements the wrapper for the data sources, a mechanism not only to facilitate access to the data sources through the mediation schema, but also to translate the user query for the separate data sources. The result of this process is a virtual, global conceptual database schema that allows the utilization of a single entry point to a multitude of databases. Both techniques can be viewed in Figure 2.3.

Another critical shortcoming of the data warehouse pertains to the addition of new data sources. On the level of logic, two data sources may be related according to either a Global-as-View (GaV) approach or a Local-as-View (LaV) approach, see [12]. GaV describes a global schema by defining views on the source schemas, whereas LaV defining views on the global schema. The advantage of LaV lies in the relative ease of adding new data sources and the

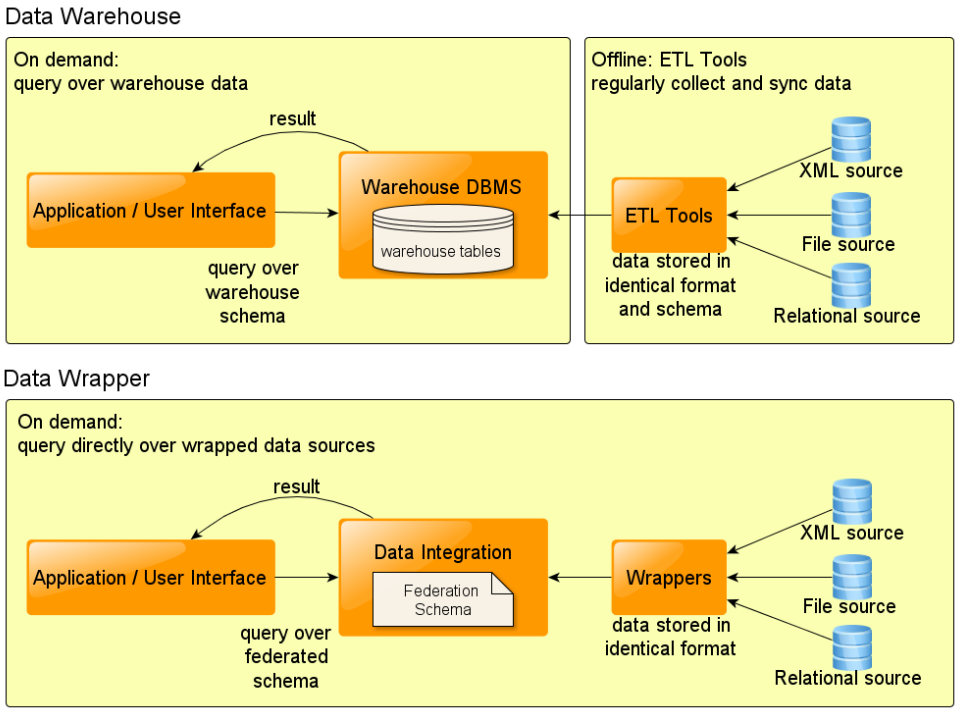


Figure 2.3: A comparison of two popular techniques for data integration - the data warehouse above, which pulls and converts data into a separate data storage, and an altered schema and the data wrapper below, which performs online queries directly on top of a live data source through it. [11]

stability of the mediated schema, while its disadvantages lie in the complexity of inferences required for resolving queries on the mediated schema. The advantages of LaV are the disadvantages of GaV, and vise versa. For a theoretical formulation of both approaches, see [9] and [13].

As new data sources are required to be added without changing a system-wide schema, the LaV approach conducted by the wrapper is preferred.

2.4 Grid and cloud computing

The vision that computing resources are to become a commodity rather than a product is not new. The birth of the concept can be dated back to the 1960s, when John McCarthy opined that 'computation may someday be organized as a public utility' [14]. A brief historical summary on this technological trend can be found in [15]. Since then Internet technologies underwent a major evolution, making this vision more of a reality.

> A hundred years ago, companies stopped generating their own power with steam engines and dynamos and plugged into the newly built electric grid. The cheap power pumped out by electric utilities didn't just change how businesses operate. It set off a chain reaction of economic and social transformations that brought the modern world into existence. Today, a similar revolution is under way. Hooked up to the Internet's global computing grid, massive information-processing plants have begun pumping data and software code into our homes and businesses. This time, it's computing that's turning into a utility. [16]

As science and engineering were facing the ever growing need for computational solutions, a computing model, called grid computing, was introduced in the 1990s. It was designed to make computational resources available as readily as electric power through the power grid and to support large scale high performance computation. The model, in coherence with utility computing, introduced the possibility to link several super-computing centers via the grid, letting researchers to use them for potentially large-scale projects on-demand. It introduced dynamic provisioning, giving an illusion of infinite supply of computing resources. [17] summarized the challenges of the grid approach. Later, the focus shifted onto linking up vast amount of data in addition to computational power.

Grid computing was developed to meet the need for high performance computation by sharing HPC resources. For example, the US based TeraGrid [18] infrastructure (between 2001 and 2011) comprised several super-computing centers interconnected through the Internet. The resulting combined computing power exceeded that of any single super-computer available. Later, grids with different specialization were developed to suite the needs of the scientific community. The EGEE Grid [19], for example, focused on management and sharing of extensive amounts of distributed data in the European research community.

Cloud computing, on the other hand, has established itself in a commercial environment. It is the result of a combined evolution of several technologies that have reached the maturity to meet commercial standards. These technologies incorporate grid computing methods as well as virtualization, distributed storage and parallel computing. The combination of these technologies produce a new flexible and scalable computing model. Some of the pioneer enterprises of cloud computing included Salesforce [20], Google [21] and Amazon [22].

Viewed from a broader perspective, cloud is a metaphor for the Internet. The concepts behind cloud computing allows for a successive outsourcing of computing resources to the Internet. Through dynamic provisioning and management of virtual instances, it provides an illusion of unlimited computing and storage resources. But the main reason for its success lies in its convenience. While traditional IT requires constant attention of developers to every aspect of the system, such as hardware, storage, networking, operating system, middleware and domain application. In a cloud these abstractions can be chosen by application requirements. The developer has the possibility to choose a subset of computing resources (via a coarse-grained layer of abstraction), delegating the remaining responsibilities to the service provider. This lays

a foundation for service-based models of cloud computing, see Figure 2.4.

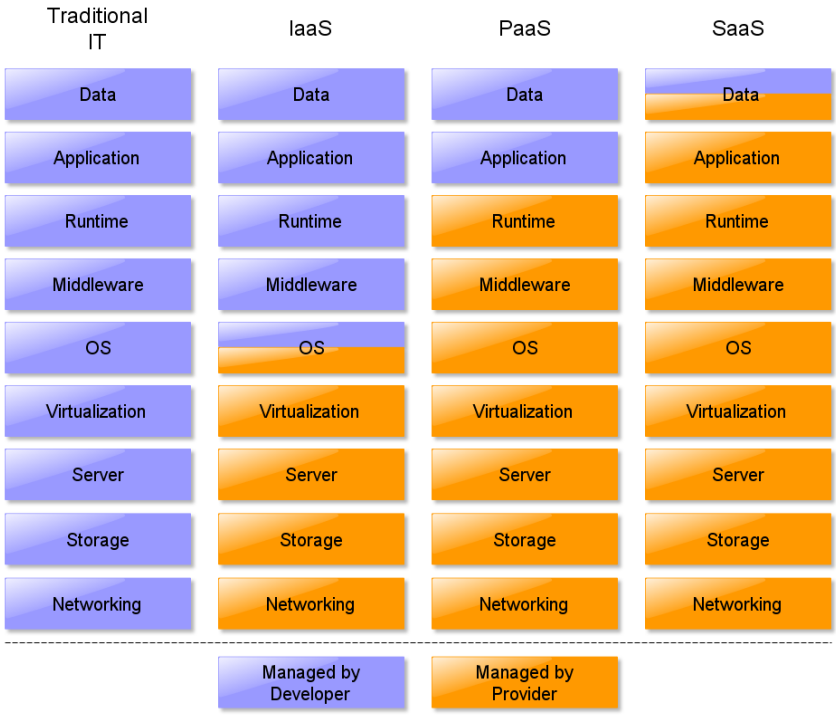


Figure 2.4: The above is a comparison of the main cloud computing models and their respective abstractions. The outsourcing of resources depends on the model, while the traditional IT approach demands the management of all available computing layers. IaaS virtualizes OS and the underlying infrastructure, PaaS provides a runtime platform in addition to the IaaS, while SaaS provides the end-user software. [23]

The Infrastructure as a Service (IaaS) model provides the basic layer of abstraction. The provider is in charge of managing the provisioning of hardware resources. At this level, the consumers have access to virtualized operating systems, to which they are free to add software supported by the operating system. The developer is relived of the responsibility of managing load, storage, network, etc. The IaaS model still requires more administration than following cloud models, but significantly less than traditional distributed application solutions.

When using the Platform as a Service (PaaS) model in addition to the IaaS, the consumer can use tools such as libraries, middleware and integrated frameworks offered by the provider to create a software. This allows the developer to focus solely on the business logic of the application, while the provider takes care of a rich hosting environment. This environment is mainly for programming and execution of the above-lying software to support its development, testing and execution. Integrated features may include monitoring, security and concurrency management, but does not include access to the underlying operating system or its file system.

Software as a Service (SaaS) model represents the highest layer of abstraction. Here, the whole service stack is managed by the service provider. Only the data may be controlled by the consumer. Data services are built on the IaaS and provided as SaaS.

Due to the growing trend in cloud computing, several other terms in this relation have been developed, including Network as a Service, Business Process as a Service, Data as a Service, Identity as a Service to name a few. These are generally referred to as XaaS (Everything as a Service).

2.5 Service oriented architecture

Cloud applications often rely on a Service oriented architecture-type architecture. It proved to be the most fitting for building applications for this type of environment, first, by encapsuling functional components into loosely coupled services and, second, by providing them with means for interoperable communication. The Organization for the Advancement of Structured Informations Standards (OASIS) [24] describes the Service oriented architecture (SoA) baseline model as follows:

> A Service-oriented Architecture is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.

A service is an autonomic entity exposed via a network that allows calling its encapsulated functionality through a standardized, generic and well-defined interface. An access to the service is exercised in consistency with the policies and constraints defined by the service description. As defined by W3C [25], services present an abstracted logical view to the underlying computing resources, such as actual programs or databases. A service is per definition stateless, although in reality most applications exposed as services do require the state to be saved. These applications may use extensions to the service implementation to enable state management.

Other properties of a full-fledged service include

- a coarse-grained granularity to achieve low dependence between loosely coupled services;
- platform independence and general interoperability, for example, clients developed in programming languages other than the one used to expose the application may be used to communicate with it;

- location transparency, exposing a well-defined interface but hiding functional application layers;
- service discovery, supporting the means for services to advertise themselves through their definitions, allowing easy consumption without prior knowledge;
- the focus on description, meaning that all communication is provided in a machine-readable format (mainly XML).

To wrap up the discussion of key features of SoA, [26] states:

> Services are well defined, self-contained modules that provide standard functionality and are independent of the state or context of other services. Services are described in a standard definition language, have a published interface, and communicate with each other requesting execution of their operations in order to collectively support a common task or process.

In view of the fact that organizations in the cloud are bound to be dynamic and ever-changing, services use a communication standard to facilitate the flow of information to the client as well as to other services. Communications are message-oriented, meaning that a request/respond message exchange pattern is realized between the service provider and consumer. Due to constant improvement of the underlying applications, services usually wrap applications according to predefined standards. Otherwise, after each update, every consumer of the service will have to adopt to the newly changed processes, which is uneconomical. To accommodate this flexibility, application are wrapped as services to a collection of autonomous and loosely coupled processes [2]. The concept stipulates a *publish-find-bind-execute* procedure resulting in the well-known SoA triangle, depicted in Figure 2.5.

To establish a link for the sequences of Figure 2.5, several protocols together represent an implementation of the SoA for the web, commonly referred to as web services.

2.6 Web services

Web services present an implementation of the SoA design pattern as self-contained applications accessible to other applications via the web through an Internet-oriented interface. They are exposed by an endpoint and communicate in through XML messages.

The implementation of web services use the following three specifications: the Simple Object Access Protocol (SOAP) [28], the WSDL [29], or Web Services Description Language, and the Universal Description Discovery and Integration (UDDI) [30]. Their descriptions can be found in the following sections. Figure 2.6 depicts their relation in the SoA triangle.

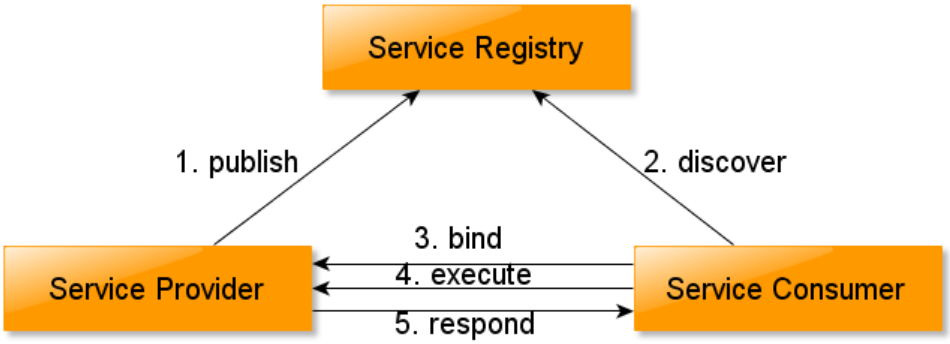


Figure 2.5: The three main components of a SoA include a service registry, which brokers a particular service between the other two parties, the service consumer and its provider. After the consumer has found a service provider that offers the desired functionality, a binding link can be established and the execution of the functionality can begin. For more information, see [27].

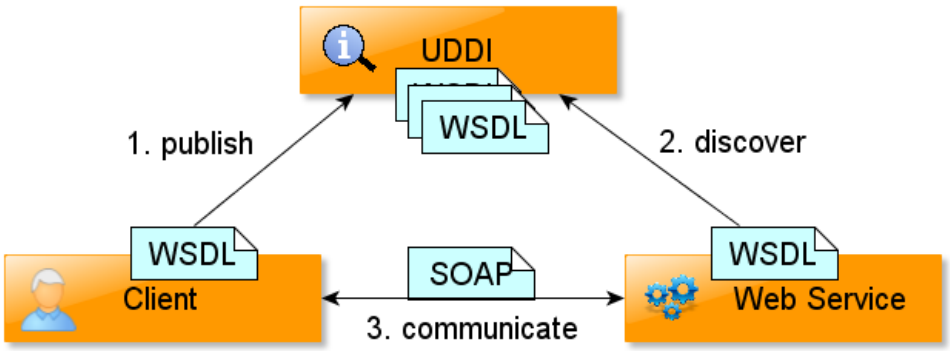


Figure 2.6: Web services present an implementation of the SoA-type architecture. The SoA triangle is specified by defining service description and communication protocols, as well as a discovery platform.

2.6.1 SOAP

SOAP is a lightweight, XML-formatted document suited for exchanging information with a web service. It contains an envelope with a head and a body. The body may contain one or more elements that describe function calls, including the specific parameters. The application-specific examples are provided in Section 6.4.

The SOAP protocol supports both a Remote Procedure Call (RPC) style and message-oriented information exchange. The RPC-style call is based on a request-response procedure. If an endpoint receives a request, it generates a corresponding response while blocking the process until it has done so. This is also known as a synchronous call. The message-oriented type call uses queues, where messages are put to the server queue and processed without blocking

of the system. The client may fetch his response at a different point of time. This technique is conducted asynchronously.

SOAP is platform independent. The messages are well-defined and machine-readable, making it possible for other web services to consume them more easily. SOAP is transfer protocol independent. Although HTTP is widely used, messages can be transmitted through different protocols that are Internet-oriented, such as FTP, SMTP or RMI/IIOP. To support attachments W3C extended SOAP with the Message Transmission Optimization Mechanism, which allows for an efficient transmission of binary data to and from web services [31].

SOAP messages can be encoded in several styles. The one used by the application is described in Section 6.4.

In sum, SOAP is a XML based protocol is an envelope containing the message content, encoding rules for expressing data types and a representation of remote procedure calls.

2.6.2 WSDL

Web Service Description Language (WSDL) defines and advertises a web service. WSDL presents the generic interface of the web service written in a machine-readable XML format. A WSDL document describes the a web service so that a client or other web service may access functions, and provides a communication format for data binding. The WSDL provides a usage description of a web service without prior knowledge of it. WSDL 2.0 is the current standard adopted by the W3C consortium. The description of a web service can be modeled in two parts.

The abstract part describes a general interface that defines the communication, in particular the exchange patterns of messages, including their sequence and cardinality. Operations define one or more messages in the pattern, while the interface combines these operations. The specific part of the description contains bindings with the transport and wire format for interface. A service endpoint associates network address with a binding. Finally, a service collates the endpoints to a common interface. Figure 2.7 shows the conceptual WSDL component model.

It is also possible to modularize the WSDL through the import directive, separating it into conceptually distinct documents. For further information on WSDL, see W3C WSDL 2.0 Recommendation [29]. The data service specific web service definition is described in Section 6.3.

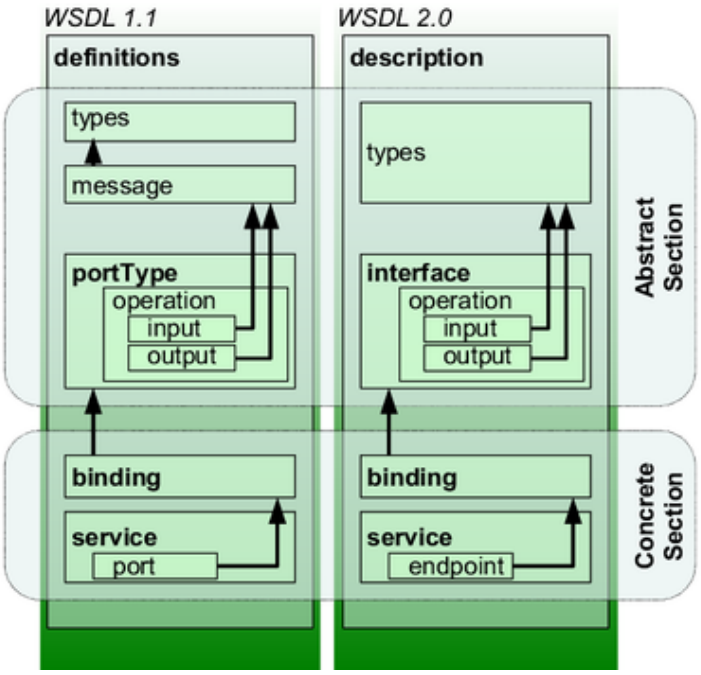


Figure 2.7: The specification of the WSDL document is divided in two parts. The abstract specifies a general interface and message exchange patterns, and a format. The concrete part allows the specifications of the web service network bindings, including its endpoint. [32]

2.6.3 UDDI

The Universal Description, Discovery and Integration service is a platform-independent registry framework for web services, similar to the yellow pages. It is the standard public implementation for publishing or discovering web services. The framework itself uses SOAP to communicate and WSDL to describe the published web services.

Although a UDDI registry might improve the usability of a web service, this technology did not establish itself as firmly as the other two standards. Major issues included difficulties in specification and implementation of it [33]. This component of the SoA triangle in Figure 2.5 is not relevant for the VPH-Share project, and will consequently not be considered here. In the VPH-Share project, a custom service registry will be provided at a later stage.

2.6.4 The WS-* protocol stack

Beyond the three main specifications, several others protocols where added as the technology evolved. They are collectively referred to as the 'WS-*'. Although adding more specifications regarding specific issues related to web services, these protocols are maintained and supported

by various, even competing organizations, and may extend as well as overlap each other. Figure 2.8 presents a web service structure including several WS-* components.

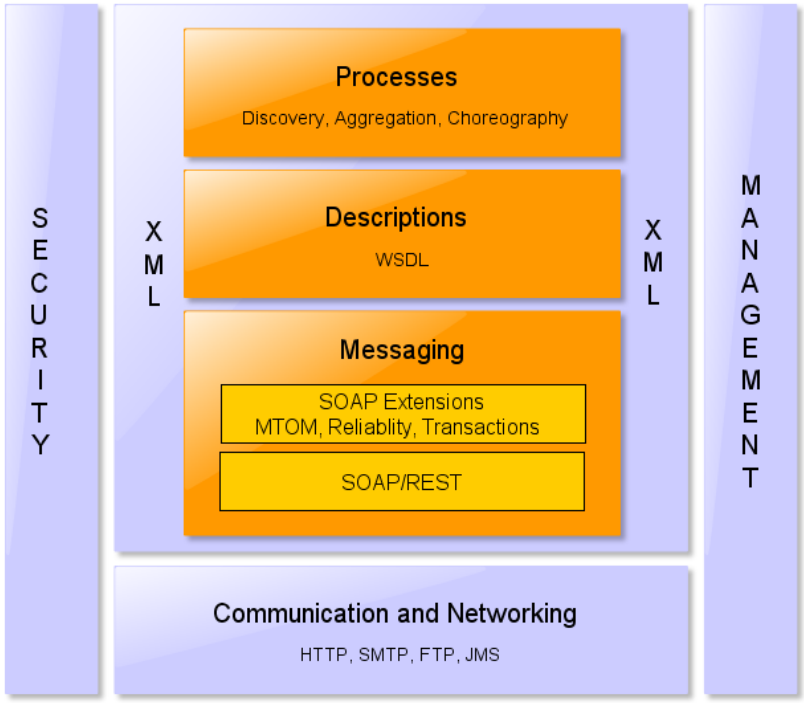


Figure 2.8: A typical web service architecture consists of the three first generation specifications: discovery, description and messaging, with additional extensions for security, management and communication. [34]

The following brief list presents a description of the main extension areas with some WS-* protocol examples, as described by Wikipedia [35].

- XML Specification - specifying the format of the XML documents, including XML, XPath, XQuery, XML Pointer, ect;
- Messaging Specification - specifying the messaging protocol, including SOAP, SOAP-over-UDP, WS-Notification, WS-Transfer, WS-Addressing, ect;
- Metadata Exchange Specification - primarily used to describe web services, including WSDL and UDDI (and some extensions to them), but also WS-Policy, WS-Discovery and WS-MetadataExchange;
- Security Specification - used to establish a secure communication, a user access control system with permission management or ensure privacy, including WS-Trust, WS-Security, XML Encryption, XML Signature, ect;

- Reliable Messaging Specifications - WS-ReliableMessaging, WS-Reliability and WS-RM Policy Assertion
- Web Services Interoperability (WS-I) Specification - guidelines that help to improve interoperability between web service implementations;
- Transaction Specifications - support for transaction operations. WS-BusinessActivity, WS-AtomicTransaction, WS-Coordination;
- Resource Specification - as pointed out in Section 2.5, a stateless web services may not be sufficient to fulfill the requirements of a particular web service. The WSRF specification falls under this category and supports stateful web services.

The WSRF, WS-I and WS-Addressing extensions are considered in this work. The WSRF, as an extension to SoA provides the specification for deploying and managing services as stateful resources. The WSRF framework encases the WS-Adressing protocol that is used to address a specific resource of a specific endpoint along with serveral other WS-* specifications, including WS-Resource, WS-BaseFaults, WS-ServiceGroup, WS-ResourceProperties and WS-ResourceLifetime. These specifications have to be taken into account for a web service to be WSRF compliant. The WS-I specification provides guidelines for developing interoperable services through profiles. The WS-I Basic profile provides guidance for the three core specifications – SOAP, WSDL and UDDI.

2.7 Summary

This chapter provided insights into the basic concepts of web services, data access and integration. It emphasizes the important role played by web services in providing inter-connectivity of data sources. The protocols for web services SOAP/WSDL as well as WSRF will play an important role in the implementation of the services in the VPH-Share project. The next chapter will provide a detailed analysis of the the VCE data services and further discuss the features of the technology described above.

Chapter 3

Data access and integration services

The belief that complex systems require armies of designers and programmers is wrong. A system that is not understood in its entirety, or at least to a significant degree of detail by a single individual, should probably not be built.
–Niklaus Wirth

This chapter describes the data access and integration services from several perspectives. The challenges in that arose during the work on project context are described with relation to the system and its environment. The remainder of the section presents a layered system architecture, including description of its components, as well as the choice of technologies implementing them. Finally, a final layered architecture of the data services is presented.

3.1 Challenges of a distributed data integration system

Several issues can be identified that are relevant to a distributed data integration system. These include:

- Heterogeneity of data platforms. Since data sources typically come in various types, the platform should be able to integrate at least the most common ones, such as relational databases, XML and Filesystem data;
- Longevity. The requirements usually outlive technical decisions. The presented solution should thus allow integrating new data-management techniques by using a standardized interface. A service must also be considered long-lasting;

- Diversity of data. The platform has to be able to deal with different types of data, including primary data and its derivatives, meta data, etc. To support differently annotated data that can have the same meaning, a semantic knowledge-based technology has to be included;

- Multiplicity. Many data sources that can be geographically distributed, independently owned and administrated. They may not have a common goal or design. A common scheme for the data, involving for example foundation types and ontologies, can be implemented only at a higher level;

- Scalability. Many data sources may contribute to the data platform. These sources contain large collections of data. The infrastructure has to take into account that many users will use the system simultaneously;

- Performance. A reasonable data throughput for multiple clients has to be provided in reasonable time;

- Fault tolerance. Distributed systems are dynamic and complex. Failure of individual resources should not lead to system failure, but to recovery of those resources;

Having outlined the challenges, appropriate environment and system components have to be specified in order to meet them. The utilization of cloud technologies will help meeting at least some of these challenges proper. The dynamic nature and flexibility of a cloud allows entities to be spawned when required, so that the scalability challenge can be met. The issue of longevity can be resolved with help of standardized interaction between the services and clients through the web service protocols. The same applies to distributed data sources, as they can remain under remote administration and be exposed as a service, in which case a wrapper is required to automatically translate changes in the data source and expose the scheme accordingly. As for diversity, heterogeneity and fault tolerance, an appropriate middleware has to be applied to resolve these issues. A discussion of those will be postponed to the following chapters.

3.2 Abstract architecture

As depicted in Figure 3.1, the composition of services can be classified in two main types: a fine-grained data service that wraps a particular data source and exposes as a web service, and a coarse-grained federation service that orchestrates data querying and federations across several data services. The federation service can only access data through the corresponding data service. A read-only access is required since data sources are to be managed remotely, outside of the infostructure. In addition to these services, the client will compose several tools that allows users to define queries for exploring the data sources.

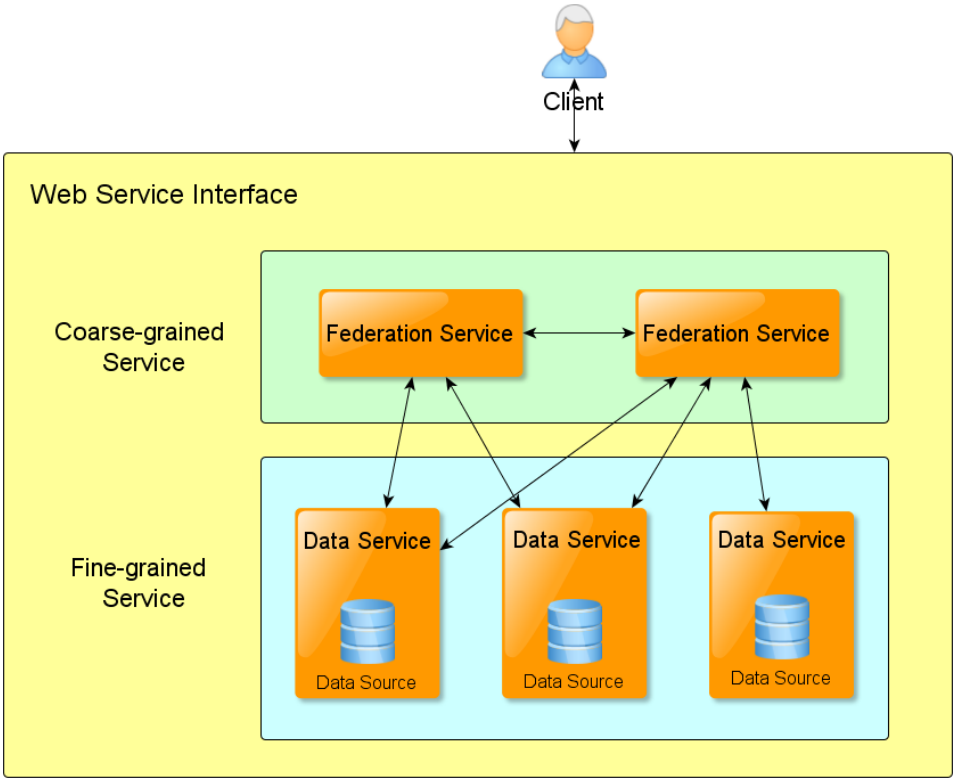


Figure 3.1: The data infrastructure comprises federation and data services. The coarse-grained federation services manages querying across several underlying fine-grained data services that expose individual data sources. Both types of services implement identical interfaces, as their access patterns are similar. The user can consume both services in same manner. Federation services may include other federation services in addition to the data services.

The services consist of a web service interface, a middleware that is able to meet the challenges and requirements for exposing and exploring data sources as web services, and a wrapper that is able to communicate with the data source. Several tools regarding a client and service administration have to be provided in order to make the services usable. These components can be collected in client, server and infrastructure groups, as visualized in Figure 3.2.

In addition to data exploration and delivery, the client-side components are responsible for providing the client API, so that users may build their own data applications in addition to the ones provided. The server-side components are responsible for the service interface and implementation, and for providing data extraction and transformation functionalities above the middleware. The bottom layer encases data storage and virtualization in a cloud infrastructure. The actual data source facilities lie scattered across the network and should be exposed via virtualized data services.

In the following, each separate component is discussed with regard to its requirements. The

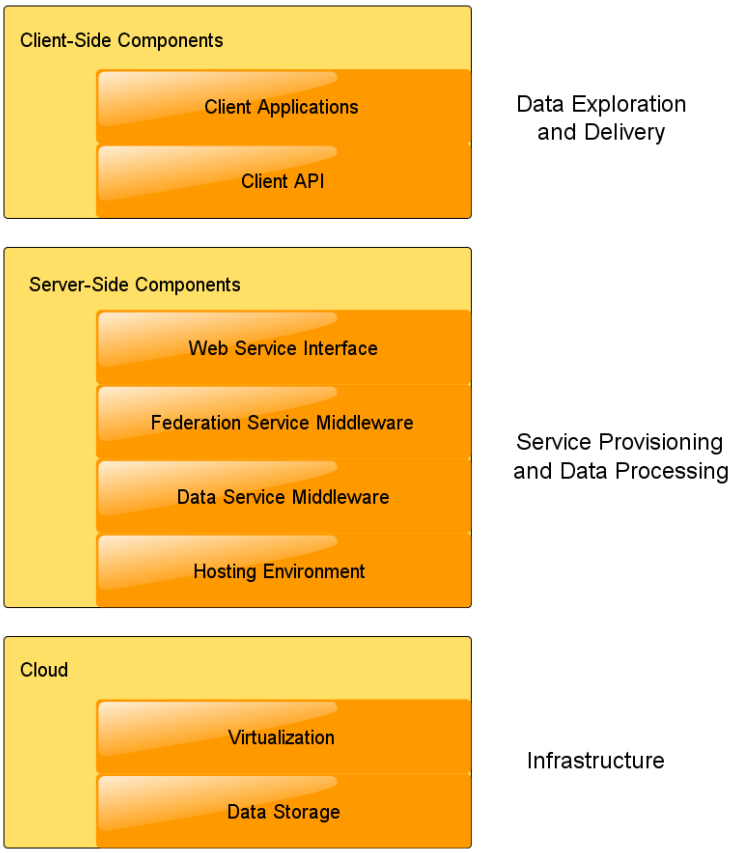


Figure 3.2: Layered architecture of system application showing the structure of the high-level components.

implementation technologies are taken into account for each layer, following a discussion of the choices made for this system.

3.3 Middleware

The middleware is the backbone of a data integration system. Not only does the middleware manage connections to the underlying data sources by using wrappers and provide mechanisms to expose it as a web service, but it must also be able to orchestrate queries across several data services, as described in the composition of the system.

The middleware receives some directive from the client with a query through the interface. If it concerns a data source exposed underneath it, the query is relayed to the wrapper, that poses the query to some DBMS. The data retrieval can begin immediately. If, on the other hand, the

query is posed to a federation service, the middleware has to reformulate and apply it to the underlying data services. Since data transmission is a costly activity, processing of it should be moved as close as possible to the originating data source.

Other functionalities should include the support for user sessions and concurrent operations, synchronous and asynchronous querying and data transformation. The middleware should also support the integration of additional data sources on-the-fly, so that the new data resources can be addressed and used in the context immediately after the deployment process is finished. The complexity of the infrastructure requires comprehensive error handling. The errors should be propagated from the source to the requester in a meaningful and standardized manner. The wrapper itself is to provide connection drivers for a particular data source.

To cope with the heterogeneity of layers, the middleware should be modular by design, so that any particular part of it could be reimplemented.

3.4 OGSA-DAI

The choice of middleware facilitating the wrapping of data sources is the main issue to be resolved. In view of the requirements several technologies come in consideration. Most of them are grid technologies:

- Storage Resource Broker (SRB) [36] is primarily a file-oriented solution that uses attributes and logical names to give access to a data resource;
- Grid Data Source Engine (G-DSE) [37] is a batch task submission system around a data source. Databases are viewed as software computing machines and as such execute tasks on top of the data source;
- IBM WebSphere Information Integrator (WSII) [38] is a commercial product from IBM. It allows federation and replication of data, as well as transformation and publishing;
- Mobius [39] is a framework for exposing and managing the sharing of data. It uses a Global Model Exchange for providing the global schema, as well as data instance management and translation services;
- Open Grid Services Architecture - Data Access and Integration (OGSA-DAI) [1] is a Java-based open source middleware for accessing data resources on the web.

The OGSA-DAI middleware is developed by the research community OMII-UK and the University of Edinburgh. It is the current de-facto standard middleware for providing access and integration of databases via web services. This middleware is used by many data-oriented

research projects, such as Admire (Advanced Data Mining and Integration Research in Europe) [40], @neurIST [41] or MESSAGE - Mobile Environmental Sensing System Across Grid Environments [42]. It has proven to be a stable, reliable and practical solution for data integration into the web. In comparison to other existing middleware, the OGSA-DAI middleware supports relational database integration, files and XML based data sources. OGSA-DAI exposes data, metadata and the schema relying on the resource's intrinsic querying mechanisms. As such, it does not require a Global Model Exchange as Mobius does. OGSA-DAI is a web service wrapper for data sources, but it can also function as a federation service due to its existing extensions, as described in the next section.

The core functions of OGSA-DAI cover the basic exposure of a data source into the web. In addition to the available components for functional extensions, various forms of querying, transformation and delivery of data are supported. OGSA-DAI's services are WSRF-complaint, meaning they will expose data sources as resources in the web. These resources manage the data sources and OGSA-DAI's functions in a similar fashion.

As intuitively suggested by the acronym, OGSA-DAI follows the Open Grid Services Architecture (OGSA). The architecture was proposed by the Global Grid Forum [43]. It defines a blueprint for building an interoperable environment for distributed heterogenous systems based on WSDL and SOAP. OGSA is a refinement of the Web Service architecture first proposed by Ian Foster in [44]. The document defines several services to comply to this environment, including services related to infrastructure, execution and resource management, data and information, etc. OGSA-DAI can implement services using a web service framework. OGSA-DAI utilizes the MTOM extension to SOAP for the transfer of data in a streamed form.

The Database Access and Integration Services (DAIS) Working Group developed the Web Service - Data Access and Integration (WS-DAI) specification of the OGSA-DAI model for integrating data sources through high interoperability within the distributed environment. The specification describes generic interfaces for representation of data sources as web resources. Specific data source types are described as extensions to the general specification in [45]. These extensions include the relational (WS-DAIR), XML (WS-DAIX), semantic (WS-DAI-RDF) and other specifications.

OGSA-DAI typically works as a set of tasks. These tasks, referred to as OGSA-DAI Activities, can be grouped by OGSA-DAI workflows; they enable data processing capabilities for the framework. Workflows can be of different types. For example, a pipeline workflow will execute activities sequentially, while a parallel workflow will so in parallel. Activities of a workflow are subdivided into three main categories: data retrieval, transformation and delivery. Workflows are able to retrieve data from all types of OGSA-DAI resources, which makes them highly customizable. This strategy allows efficient workflows compositions to be executed on the server side using OGSA-DAIs pipe implementations. The workflow may then be executed either synchronously or asynchronously. While the resources will be defined in terms of services in the implementation, workflows can be used to compose the client-side API of the VCE to provide

flexible data processing mechanisms.

OGSA-DAI provides several extension points that can reimplemented and switched to if required. These include:

- Activities for the composition of workflows and data processing functionalities;
- Data resources for different data sources;
- Presentation layers for different access interfaces to the services;
- Security for different security protocols.

3.4.1 OGSA-DAI Extensions

OGSA-DAI extension packages provide additional functionality to the core components through the data resource extension point. These usually include data resource or activity implementations for a specific purpose. The relevant extensions: the Distributed Query Processing (DQP) Engine, the SPARQL-RDF extension and the OGSA-DAI Views.

OGSA-DQP

By allowing data access and integration via Web services, OGSA-DAI integrates a service-based distributed query processing (DQP) engine [46]. The DQP extension makes it possible to federate queries and control data delivery across several distributed datasets. DQP is able to execute queries in parallel over OGSA-DAI data services and its resources in an highly optimized manner. It provides a basis for the virtual data source approach used in the VPH-Share project.

OGSA-DAI utilizes data integration and access components, allowing the system to execute user queries onto several distributed data sources. The main interaction point, the DQP coordinator, is composed of a compiler, evaluator, partitioner, optimiser and a scheduler. Together they manage the execution of a query. The coordinator parses the query, gathers the metadata from the distributed data nodes and composes a query plan using the linear left deep tree decomposition approach. The plan is partitioned, evaluated and optimized before finally being applied to a remote data source service. The evaluation of individually partitioned queries occurs at the leaf nodes, and propagated up the tree, until the point where the root leaf is processed is reached and the query execution is complete. This process is visualized in Figure 3.3.

The main advantage of OGSA-DQP is that remote data sources can be managed separately, as schemas related to the database are extracted and evaluated at the time of query execution. While the DQP component requires some additional setup configuration, it can be used in a

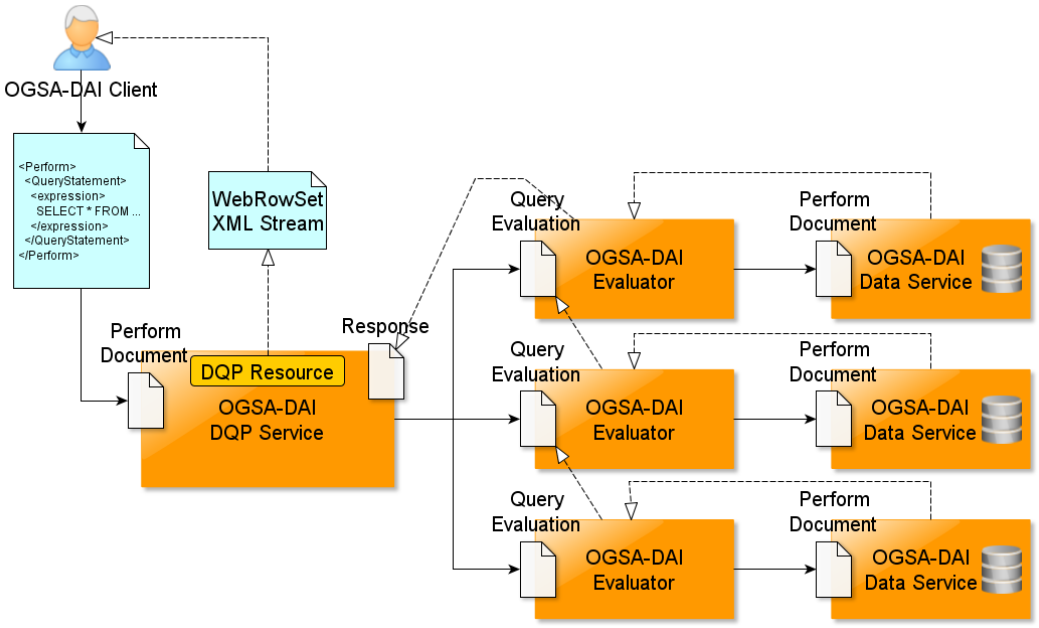


Figure 3.3: An example of a request evaluation and processing through the Distributed Query Processing (DQP) Engine of OGSA-DAI. In this example only the evaluator decomposes the query and sends them to the appropriate data service. [47]

similar manner as a default OGSA-DAI data service with regard to the locations of the remote data sources. It is registered as a resource and accepts SQL statements.

One drawback, however, is that the client has to be aware of the data distribution and integration. For instance, at attempt to execute the following query

```
1 SELECT myTable.name
2   FROM myTable, myTable2
3   WHERE myTable1.id=myTable2.id;
```

would fail, because DQP would not know which resource it should, since the two tables may not come from the same data source. Even so, they could be exposed by separate data resources. If there exists a mapping between the data resources and the tables, where for example each resources exposes one table, then the following query is correct:

```
1 SELECT ResourceID_myTable.name
2   FROM ResourceID_myTable, ResourceID2_myTable2
3   WHERE ResourceID_myTable.id=ResourceID2_myTable2.id;
```

This query explicitly annotates the mapping the OGSA-DAI resource to the dataset. This poses a serious inconvenience to the user, requiring the client to know the mapping of the resources and their corresponding tables. This issue cannot be easily resolved automatically due

to the semantics of column names, different datatypes and schemas. To address this problem, OGSA-Views is used to provide a global virtual data schema.

The basic OGSA-DQP implementation supports a subset of the SQL 92 grammar. However, due to its modular design, it enables the utilization of custom components, including grammars, but also compilers, evaluators and optimizers. For more information on the topic, see [47].

OGSA-Views

A native attempt to resolve the issue of having the user map OGSA-DAI resource names to the corresponding tables uses the OGSA-DAI Views extension. As any other extension, OGSA-DAI Views deploys an access resource to the service and wraps the DQP resource with an SQL statement. OGSA-DAI Views function in a similar manner as database views.

Essentially, a OGSA-DAI View is virtual table specifies a result set in terms of a predefined query, similar to SQL Views. The OGSA-DAI Views extension allows a view to be created on top of any resource capable of executing SQL queries. The view then exposes a virtual relational resource with its schema, which maps table names and queries. Similar to the DQP, a View rewrites the query on the abstract syntax tree level. The result is a virtual table across several data sources.

Views offer a number of benefits. Similar to SQL views, they can be used to simplify the composition of user queries. Taking the example of a query described in the previous chapter, if *myView* wraps the DQP-style SQL statement, than

```
1 SELECT * FROM myView;
```

is equivalent to the wrapped statement.

Further uses of OGSA-DAI Views include restricting data retrieval by hiding the DQP resources and allowing access through Views resources only, and combining data sources with conflicting schemas and namespaces through SQL. This functionality can be a simple renaming of column names or more complex value-replacing joins.

OGSA-DAI-RDF

The Resource Description Framework or RDF is commonly used to store metadata and information in semantic web applications. Since the processing of such is a requirement by the VPH-Share project, they will need to be integrated by the infrastructure. SPARQL, the SPARQL Protocol and RDF Query Language, is an accepted standard language for querying RDF databases.

The RDF Extension of OGSA-DAI extends it with a resource that is capable of accepting and processing SPARQL queries from the same interface. The sources it can connect to are either databases containing RDF triplets or SPARQL endpoints. While the resource wraps the

data source and exposes it as a service, activities are used to process and retrieve data from them. The result is returned in form of data tuples from the RDF graph. For more information on the topic see [48].

A more appealing idea is to use a federated approach to pose SPARQL queries, which is planned to be implemented in future releases of this software. The work will be related to [49].

3.5 Client

To support the construction of client-driven applications, the client will contain mechanisms for formulating custom application logic in terms of data fetching, transformation and delivery types that can be wrapped as workflows. As such, it can be viewed as a framework with libraries and tools that allow the customization of the whole process as a high-level client API. The client is able to approach both specific data and federation services, and will communicate in a standardized, web-based manner, constituting a front-end basis that may be replaced by any other entity communicating in the same manner. It has to formulate requests and, due to the transfer of large amounts of data, get a response in an streamed form. Finally, the client should provide a simplified querying interface, while the complex layout of the underlying data sources remains hidden.

The high-level interface of the client allows selecting the service endpoint, formulating a query, the query type (SQL/SPARQL) and the preferred mode of communication (SOAP/REST). On this layer it takes a very short code to get data from the services, it does not require prior knowledge of OGSA-DAI or any other components of the system.

The client interface builds up on the OGSA-DAI Client-toolkit. The toolkit is a more sophisticated piece of software that allows custom usage of the data services. When interacting with this layer, a user can define the whole OGSA-DAI workflow, including querying classes, transformation, delivery and other activities. The user has to be aware of the service endpoint, as well as the resource he is trying to contact. This layer provides mechanisms for building custom application atop the default client. To communicate with other data services, each deployed web service will have the client-toolkit at its disposal. It communicates mainly by exchanging 'Perform' and 'Respond' documents. The Toolkit includes an API, but can also be used through Command-Line Tools for Client-Server interactions.

3.6 Interface

The interface provides a web service endpoint through which all communication with that particular service takes place. As described within Section 2.5, the classical interface uses SOAP

and WSDL specifications to facilitate the consumption of a web service.

With the emergence of the Web API [50], the emphasis shifted from the more complicated SOAP to the more direct REST communication [7] – a natural fit for managing resource states through the basic HTTP operations. It is more lightweight, as it does not use the extra XML markup for communications and is usable through a simple browser. It is easily to build, as no toolkits are required. The main issue is that a service with a RESTful interface has no mechanisms to describe itself. Clients should know what to send and what to expect in return.

SOAP messaging still remains a widely used standard in enterprises and organizations. It provides a mechanism for services to describe themselves to clients through WSDL, which makes them more easily consumable. SOAP messages are more consistent since they adhere to a contract and follow strict type checking. Development tools support integration of existing systems into the software-oriented architecture by either generating the WSDL document on top of an application, or use an existing WSDL to generate the related source code. SOAP supports several extensions that were described in Section 2.6.4, as should the interface. Through the application of the WSRF model data sources are exposed as resources. It allows the states to be saved for each request.

The present work resolves around implementing data services using SOAP messaging, while the VCE, mentioned in Section 2.2, will use both communication techniques.

3.6.1 JAX-WS specification

The Java API for XML Web Services (JAX-WS) [51] is a standard and a reference implementation that provides a high-level API for creating SOAP-based web services in Java. As part of the Java Enterprise Edition (JEE) platform, it uses Java Annotations specified within Java Standard Edition 5 (Java SE 5). This simplifies the development and enhances platform independence for Java applications. It is the successor of the JAX-RPC standard, making the messaging model more document-oriented as opposed to relying on remote procedure call-type communication. JAX-WS complaint services benefit from a dynamic proxy mechanism that provides a formal delegation model through an exchangeable provider. A special runtime environment, included in the JRE, the JAX-WS runtime provides a means of executing a web service.

The interface includes specifications of web service operations and a class that implements them. A web service framework, supporting JAX-WS, can then integrate annotated classes into the framework and expose them as web services through the interface. The operations must correspond to the web service defining the WSDL document.

The annotations *@java.jws.WebService* and *@java.jws.WebMethod* provide basic methods for defining server-side web services. The first is used for annotating the web service class, whereas the second is used to annotate the web service methods. Other annotations, such as

@Resource can be used to inject class instances into the specified variable, one example being the *WebServiceContext* instance through which a developer can extract separate parts of a request SOAP message among other tasks. These annotations may contain further parameters of a web service, for example the namespace of the service, location of the WSDL document or the endpoint URL. For implementation of this, see Section 6.5.

The JAX-RS is the respective specification for REST-based services. As does JAX-WS, JAX-RS provides Java annotations to create services that communicate using the REST pattern. The *@java.ws.rs.Path* annotation maps the URL to the service class, while *@GET, @POST, @DELETE, @UPDATE* define the request methods used for requests.

3.6.2 Web service framework

The emergence of web service technologies has led to the development of a number of web service frameworks. The three most popular frameworks are: Apache Axis2, Apache CXF and Glassfish Metro. The following reference provides a brief summary of their features:

- Apache Axis2 is a sequel to the popular Apache Axis framework. It was developed from scratch. Its modular design makes Axis2 flexible but rather complex due to its numerous configuration and integration options. It supports a wide range of Data Bindings. Yet, Axis2 does no yet fully support the JAX-WS specification, while also having a poor scaling performance (see below);

- Glassfish Metro is the standard web service solution by Oracle. It is the reference implementation of JAX-WS, and the data object binding standard JAXB. Deployment of applications requires Glassfish App Servers, although with a bit of effort the application can be packed into a WAR-container to run on, for instance, Apache Tomcat. The main strength of this framework is its ability to integrate in the NetBeans environment and support the interoperability with the .NET framework. Glassfish Metro supports most of the WS-* protocl stack standards. A disadvantage is its less modular design, which limits its extensibility;

- Apache CXF is a fusion of the Xfire and Celtix projects. CXF is primarily designed for integration into other systems, which is reflected in the use of CXF API and Spring framework. It fully supports the JAX-WS standard, and the JAX-RS RESTful service standard, explained in Section 3.6.1. The architecture is similar to the Spring-MVC. Its extensibility owes to a Spring-Bus, making Spring-Beans modules extensible and easy to integrate. Its deployment is mainly container-based. CXF also provides several code generating tools for fast development of web services. On drawback of Apache CXF is its relatively limited support for data binding.

No framework is clearly superior to all others. All frameworks are user-friendly with respect to Java development, are frequently updated, well documented and free open-source technologies with commercial relevance. They differ only marginally in their architecture, and support for data bindings and WS-* specifications. A detailed comparison of the frameworks is provided in [52]. All frameworks support Contract First (definition of WSDL interface followed by its implementation) and Code First (implementation through POJOs followed by a contract through Java annotations), which simplifies both the integration of legacy applications into a web service as well as development from scratch.

The latest SOAP-based release of OGSA-DAI implements web services based on an outdated web service technology. The reasons for upgrading these web services are discussed in Chapter 4.

3.7 The preferred architecture

Having specified all the required layers and components, a data service system architecture can finally be composed. The layered design in Figure 3.2 is remodeled with the consideration of all the parts into a high-level architecture of the data service application depicted in Figure 3.4.

The layers contain several tiers. The 'Cloud'-tier is the hosting environment for virtual instances and contains the location of the data storage. The data source is accessible through some remote, possibly private, network.

The 'VCE Service Provisioning' environment will not only manage the runtime, but also serve as a utility tier for logging, service configuration and deployment.

The 'Data Access and Integration Service'-tier is subdivided into the middleware, consisting of OGSA-DAI and its extensions, and the Presentation Layer, consisting of the web service interface and its implementation. OGSA-DAI middleware will be extended through OGSA-DAI DQP for distributed query processing, and OGSA-DAI Views to provide a more easily usable DQP. OGSA-DAI will also be extended through several data related resources, such as RDF for semantic access, relational, XML and file for these types of data sources.

The 'Client API' tier provides the OGSA-DAI toolkit for client applications. It is able to construct OGSA-DAI workflows and specify additional parameters for the service call. Below the toolkit, a messaging component of the web service framework deals with creating, sending, receiving and processing incoming or outgoing SOAP messages, and is equivalent to the web service implementation of the presentation layer of the data service.

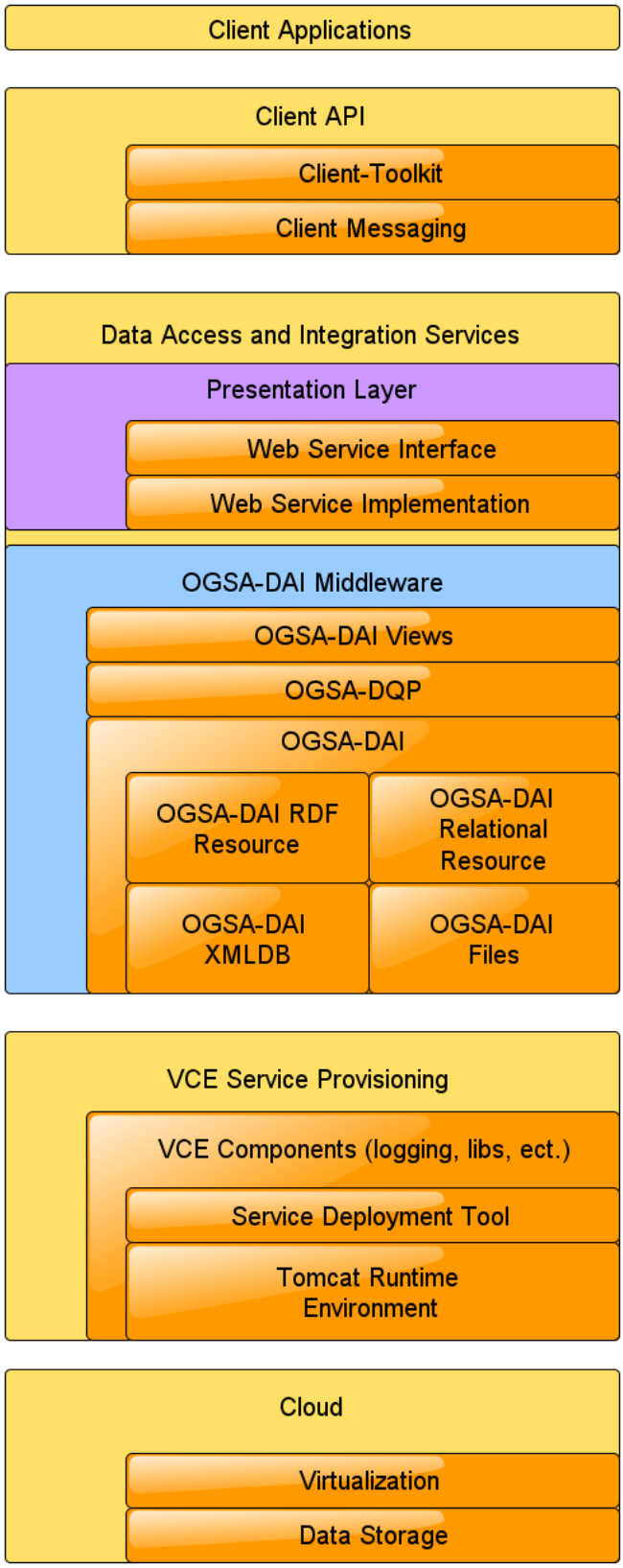


Figure 3.4: The final layered architecture design of the data services is structured into several coarse-grained layers. Each layer is composed of certain components, implementing functionality for it.

3.8 Summary

This chapter provided the context and its challenges, resulting in the basic abstract components needed for the desired data infrastructure application. The components, mainly the middleware and the interface, were discussed further in detail with relation to the current standard. The remainder of the thesis describes the need and the development process to implement a state-of-art presentation layer for OGSA-DAI.

Chapter 4

Planning and requirements

The most important single aspect of software development
is to be clear about what you are trying to build.
–Bjarne Stroustrup

Based on additional specifications for data services, this focus of this work will shift towards the implementation of a new presentation layer for OGSA-DAI using the Apache CXF web service framework. As the implementation of the interface plays an important role in the integration in the VCE, several aspects with regard to it will be discussed. Finally, the use cases for the implementation are identified and testing scenarios presented.

4.1 Purpose and scope

The data infrastructure software comprises only a small part of the VPH-Share project's complex system, but its the most important one, as all other components depend on the data that it delivers. To summarize the context of the data services, the infrastructure will comprise of several data and federation services combining large data domains in a unified access point. A customized set of such services forms the basis for an on-demand dataspace.

To recapitulate, the main objectives of data services in the VPH-Project are:

- Expose data sources as services. Data sources must be virtualized and made accessible through web services;
- Enable the search and retrieval of data through relational concepts with the addition to format data;

- Provide of data services in a cloud environment and support for deployment of data sources as web services;
- Support semantic access mechanisms used to search in RDF data collections.

The application should be modular by design. The modularity allows accommodating and integrating different types of data sources that can be extended in the future. These types can be of relational nature (MySQL, IBM DB2, Oracle, PostreSQL, Microsoft SQL Server), XML (eXist) and file storage systems (for Unix, Linux and Windows). The system should be capable of transforming (CSV and XML) and delivering (streamed download) data.

The data services will support the classic SQL queries and simple structured queries based on ontological terms stored as RDF [53] triplets. SPARQL will be used for the semantic querying. The addressed data sources can either be queried directly though their respective exposing service, or through a federated service. Data sources must be in the form of web service resources. The data service interface will adhere to the VCE, provide a high-level API, a client toolkit and other utilities for the data services.

4.1.1 A new OGSA-DAI presentation layer

As suggested in Section 3.6.2, the OGSA-DAI's SOAP-based presentation layer is outdated. It latest version was released in April, 2006, and is still based on the Apache Axis 1.4 (JAX-RPC based) web service framework. The inadequate support for current state-of-art WS-* protocol extensions and the lack of compliancy with the newer standards makes the implementation rather impractical, while its low performance creates a bottleneck for the computational workflow of other VPH-Share computational services.

The data services require a new SOAP-based web service layer that will be implemented for support of the next generation of web service technologies, including the JAX-WS, SOAP 1.2, WSDL 2.0 and other WS-* specifications. The fact that VCE requires the data services to be accessed via a RESTful pattern is another reason for requiring an upgrade, as the newer frameworks supports both JAX-WS and JAX-RS specifications.

To choose the appropriate web service framework among the ones discussed in Section 3.6.2, several external performance tests were observed in [54] and [55], including that of the OGSA-DAI team [56]. These show that Axis2 performs the worst when dealing with large amounts of clients and data (scalability problem).

These external evaluations as well as the architecture model and features suggest that the Apache CXF web service framework meets requirements in terms of usability, integration support and performance. It provides a high-level API to create web services. The integration of

existing software is simpler than on Axis2 or Glassfish Metro because former is based on the Spring framework [57].

A software release will include the data service release within the VCE, including the Java code and the Java docs for the described services.

4.2 Users

The following main user classes and their actions can be identified:

- Service administrators, who configure and deploy a service on a data source;
- Client users, who query a federated or direct resource to retrieve data;
- Applications or services, who extract data needed for further processing.

In addition to the three types above, there are developers, who build applications on a high-level client API.

Although humans may use the client to query a data source manually, it is assumed that most requests will come from other web services requiring the data for further processing and analysis. This scenario requires a strictly web service compliant interface, which is another reason for relying primarily on SOAP-based services instead of RESTful-based services.

4.3 Constraints

The development of the presentation layer follows the guidelines:

- The new interface must conform to the WSDL definitions provided by OGSA-DAI;
- The application must be deployable on the Apache Tomcat application server;
- The presentation layer implementation must conform to OGSA-DAI codebase;
- The coding style must adhere to the guidelines described in [58]. Crucial for maintainability, the code of the presentation layer should be consistent to the rest of OGSA-DAI's code;
- The implementation must not compromise the existing functionality of OGSA-DAI;

- The new service should retain the functionality of the old one. Changes in functionality should be kept to a minimum.

4.4 Boundaries

Several aspects that will not be discussed further in this thesis, but still concern the data services and the interface, are mentioned below:

- The cloud infrastructure (IaaS) will be provided by partners of the VPH-Share project, see [5]. This application should be delivered as a virtual appliance. This means, it should be possible to provide a preconfigured virtual machine with a set up runtime environment for the software to run 'out-of-the-box'. This functionality is by the VCE components;
- Dealing with clinical data in the context of the VPH-Share project poses a delicate matter, as the security of data is a top priority. This work, however, assumes that all communications is conducted through a secure network and that the exposed clinical data is de-identified, as is planned by VPH-Share;
- The publishing and de-identification of data takes place in a separate suite (work package 3 of the VPH-Share project). The data services deal with connecting to and exposing of a prepared data source;
- It is intended to provide querying access to a semantic data server only, i.e. semantic logic mechanisms on top of the SPARQL-RDF endpoint. The corresponding functionality is planned to be implemented within work package 4 of the VPH-Share project;
- It is not intended to automatically create mappings for the federated schema. The service administrators must configure the DQP and Views so as to provide users with an easy access to the distributed data sources.

4.5 Presentation layer requirements

The following is a list of functional requirements for the presentation layer of OGSA-DAI:

- The presentation layer is required to implement every service specified within the OGSA-DAI WSDL definitions;
- It has to implement every operation defined within the services;
- It has to implement data bindings for the operation parameters in form of complex data types (Java beans). This includes the handling of input and output stream types, Base64 encoding/decoding and object serialization.
- The same applies to operation faults and exceptions;
- The presentation layer has to implement SOAP message processing capabilities, in order to define custom SOAP header directives;
- The presentation layer has to be WSRF and WS-Addressing-compliant;
- The server-side implementation will invoke core functions through OGSA-DAI's functional resources;
- The presentation layer has initialize the OGSA-DAI context as soon as the web service application has been started;
- Binary data transfer will use the MTOM extension of SOAP.

The details of the OGSA-DAI architecture in relation to the presentation layer are further discussed in Section 5.1. There, a more detailed description about how services are to be mapped to functional resources and what data is exchanged between client and server is given.

4.6 VCE integration

The VCE will encase the data service implementation to provide utilities for a higher level service. In addition to SOAP messaging, the VCE will expose a RESTful interface for querying and data exploration. The client will then use its own interface, in which the user can choose the type of preferred communication (REST/SOAP), the type of query (SQL/SPARQL), the service endpoint (direct or federated), and the query itself. The result of a successful query request are saved in a user specified file.

In addition, the VCE service deployment tool provides a graphical and a command-line interface for automated service deployment. This tool, comprising Apache Ant tasks, manages the set up of the necessary runtime environment, the configuration of the OGSA-DAI's data resources and other service configuration parameters. The tool then creates a web application in the server container and copies all necessary files to it. This includes configuration files for logging, as well as web pages and libraries related, for example, to OGSA-DAI and Apache CXF. The Service Deployment Tool will be covered only briefly in the implementation, as it is not the focus of the present work.

The software release for providing generic construction of data services is planned to be available in the VPH-Share application cloud.

4.7 Use cases

The use cases can be categorized into two main groups. The first relates to the usage of the services. A user of the system, who may either be a human or a federation service, could select a data service, query and retrieve the desired data. Transformation and delivery are optional, but are closely connected to the data extraction functionality of the system. The query is generalized by either relational (SQL) or semantic (SPARQL) types. When a federation service is called, it should manage the collection of data from other data services, while storing preliminary results for further filtering.

The second category is related to the administration and deployment of a service. Here the necessary connection configuration has to be provided for the services to work properly.

Two primary use cases for the data services can be identified: 'query a data resource' and 'query a federated data resource'. They play a central role in the purpose of the data services and will be described in detail below.

Table 4.1: Primary use case descriptions

Use case	query a data resource
Input	query, query type (SQL/SPARQL), communication type (SOAP/REST), service endpoint
Output	streamed retrieval of data from the data source
Application flow	A user formulates a query and configures the parameters for the request. The client toolkit constructs a workflow consisting of activities for data querying, transformation and delivery. The client-side presentation layer classes construct a SOAP request message. The message is sent via the web. It is then processed by the server, returning the resulting data via a stream in the desired format.

Alternative 1	A user utilizes the client toolkit to construct a custom workflow directly. The workflow consists of different transformation or delivery activities. The user then manually invokes presentation layer proxies to send the request and process the result.
Alternative 2	The data service is queried by a federation service as part of a distributed query processing procedure. It requests the data schema of this service before the actual querying can take place. Only raw data is transmitted to the temporary store of the federation service.
Faults	Faults of this use-case are either • OGSA-DAI related configuration errors pertaining to data resources, unknown resources, workflow and activity faults, compatibility and OGSA-DAI configuration faults, or IO exceptions; • Service related errors, such as endpoint not reachable, WSDL contract faults, or message corruption and parsing faults; • Data source drivers errors, such as no connection to the configured data source, schema related faults, or non-existent data querying.

Use case	query a federated data resource
Input	federated query, query type (SQL), communication type (SOAP/REST), service endpoint
Output	streamed retrieval of data from multiple data sources
Application flow	A user formulates a query for a view resource and configures the parameters for the request. The client toolkit constructs a workflow and sends the request to the service. The service then processes the request and retrieves the schemas from all data services in question. A temporary data storage is initialized. The federation service constructs a query plan for each data service and uses the same client toolkit to send requests out. The underlying data sources deliver the data to the temporary storage, where it is processed further if specified to do so, before returning it to the client.
Alternative 1	A user formulates a query directly for the DQP resource instead of a view, along with other service parameters.
Faults	Faults of this use-case are either • Operation faults propagate from the bottom up. A DQP service only adds incompatible requested schema faults and temporary storage-related faults.

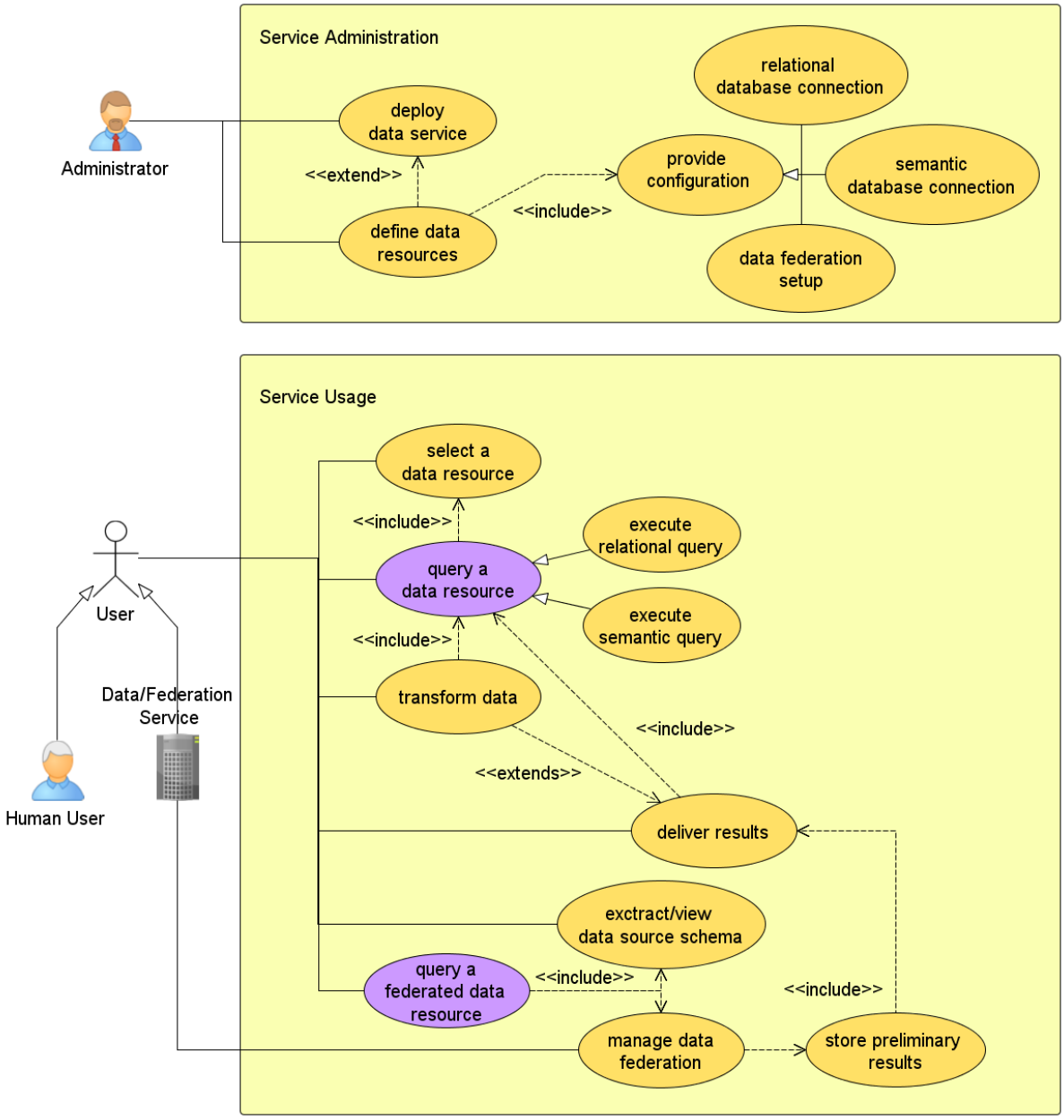


Figure 4.1: This use case diagram provides a basic overview of the data service functionality from a user's point of view. The main categories split into administrative tasks and the actual service use. Two use cases are considered a priority for the data services: 'query a data resource' and 'query a federated data resource'.

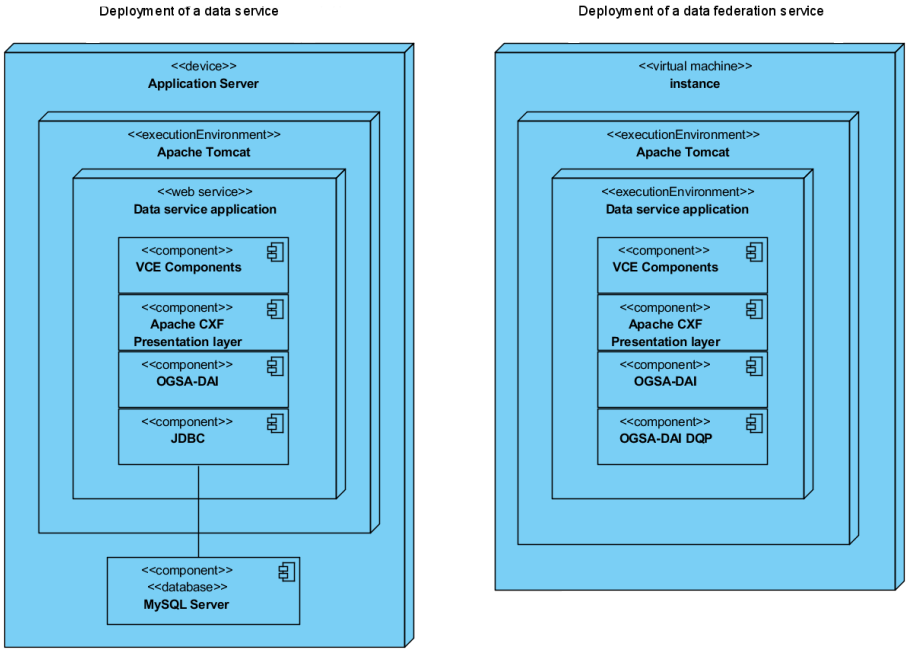


Figure 4.2: This diagram provides an overview of the evaluated deployment set up. Direct data source services will be deployed on physical machines with direct connection to their data sources, while federation services will be deployed as virtual data appliances for a cloud.

4.8 Deployment scenarios and testing

The deployment scenario of the data services will consist of two types. The first is the deployment of the data services that directly expose the data sources. These services have a direct connection to the MySQL server running locally on a physical machine. The second type regarding federation services that will be deployed on virtual machines as virtual data appliances. The deployment diagram in Figure 4.2 depicts this setup.

As mentioned in Section 2.1, the VPH-Share project will provide the main application scenario for the data services. Several real-life databases were provided for testing. These databases stem from clinical institutions, and are prepared to meet the privacy and security standards (de-identification). In addition to these data, data will be generated with a similar schema to provide larger sets for testing. Three different data sets should provide a first assessment of system performance. They are described in detail in Chapter 7.

The primary tests will compare OGSA-DAI with Apache Axis presentation layer to Apache CXF implementations in terms of data fetching time. Unit tests focus on the time till the final result is received by the client. Both, separate data sources and federation services of different sizes will be tested. The test should thus also expose the overhead of the federation in both implementations. The comparison between implementations will be carried out on locally

deployed services.

The second scenario will demonstrate the flexibility and performance of the data services and in a cloud environment. The data services that expose data sources directly will remain locally in a private cloud, whereas the federation service is either deployed in the infrastructure cloud of the VPH-Share partner in Poland, or on Amazon EC2 instances. In sum, the tests will compare federation performance between services deployed on a local cloud for reference, a remote cloud and a commercial cloud. Figure 4.3 summarizes the second testing scenario.

4.9 Summary

This chapter provided the requirements for the design and implementation of the Apache CXF presentation layer for OGSA-DAI. User classes for the services were identified; constraints, boundaries and implementation requirements set. Following this, some use cases were presented and the two primary ones described. The final chapter presents the deployment and testing strategy for the presentation layer and the data services.

The following chapter addresses the design of the implementation. In particular, it explained exactly how OGSA-DAI communicates through web services and what classes are needed through a class diagram. The sequence diagrams provide the application flow for the primary use cases presented in this section.

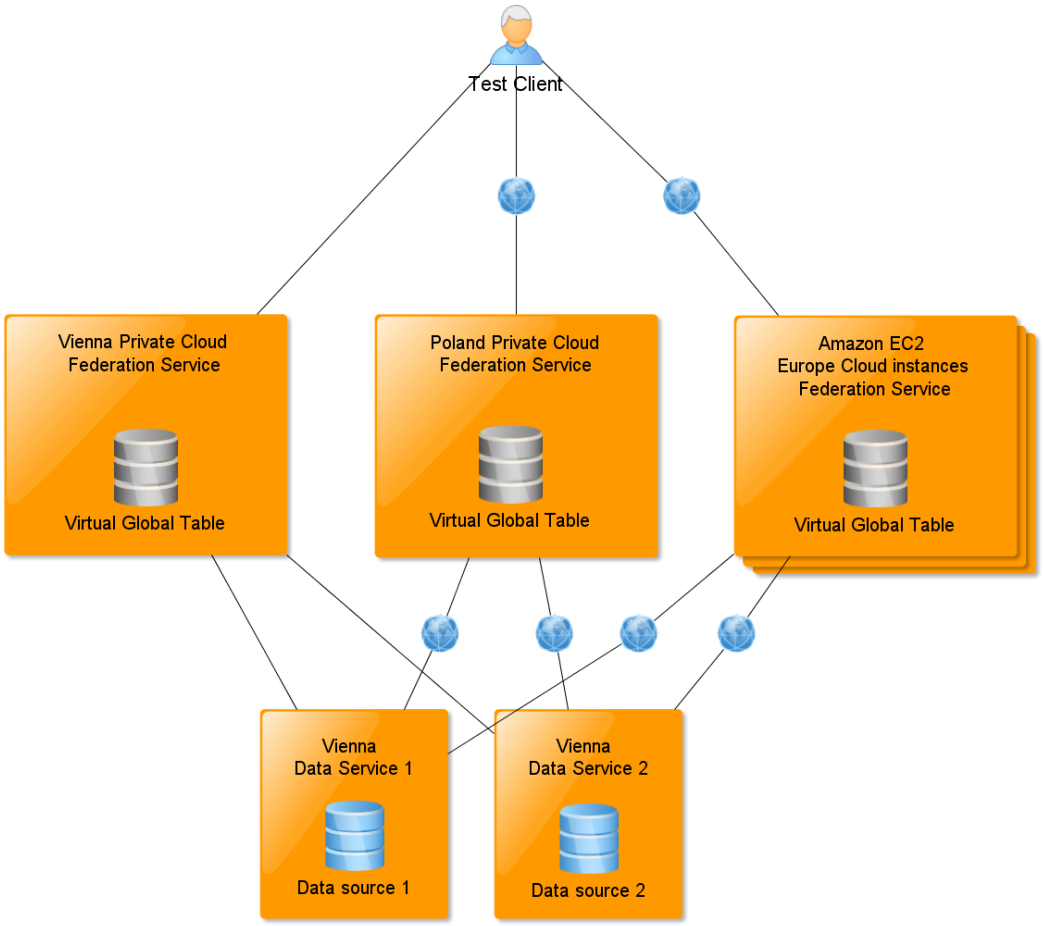


Figure 4.3: This scenario is designed to test the flexibility and performance of the services in a cloud environment. In this scenario, the data services remain fixed in a private cloud, while the federation services are deployed on several clouds. The Vienna federation service will serve as a reference, as it is placed in the same cloud as the data services. The performance will be tested in relation to other remotely deployed federation services, including those in the Amazon EC2 cloud.

Chapter 5

Design and modeling

We try to solve the problem by rushing through the design process so that enough time is left at the end of the project to uncover the errors that were made because we rushed through the design process.
–Glenford Myers

This chapter describes the architecture of OSGA-DAI in relation to the presentation layer and the specification of the data exchange for the service operations. The next chapters provide sequence diagrams for the primary use cases, and a class diagram outlining the modeling of the presentation layer. The final section describes the integration of the data services into the VCE, before proceeding into the implementation in the next chapter.

5.1 OGSA-DAI architecture

Chapter 3 provides the description of the functional capabilities and elements of OGSA-DAI. Here, an architectural view of the OGSA-DAI middleware, and, in particular, the presentation layer shall be provided.

OGSA-DAI is based on the Model-View-Controller pattern. Each layer of the pattern is based on a modular composition and is therefore extensible. The Model layer comprises of functionalities for accessing heterogeneous data sources in compliance with the WSRF. The Controller layer embodies the business logic of the middleware, and provides an execution engine of OGSA-DAI workflows and activities. These functional components are also represented as resources. The View, or the presentation layer of OGSA-DAI, is represented by web services for client interaction. The high-level layered OGSA-DAI architecture in Figure 5.1 depicts the server-side three-tier architecture.

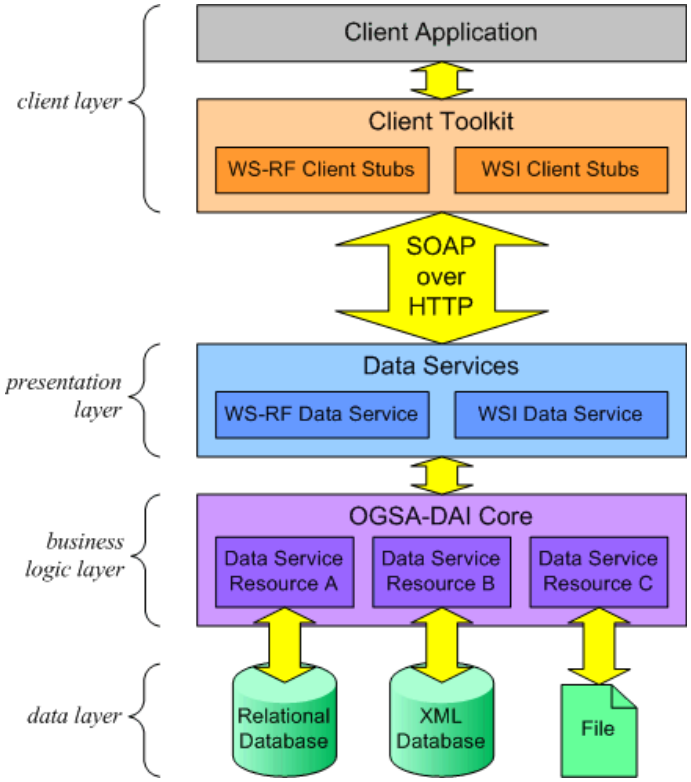


Figure 5.1: The High-level Architecture of OGSA-DAI layers.

The presentation layer can be a WSRF or a WS-I implementation. The WSRF implementation is fully compliant with the WSRF standard, but is mainly designed to be used for a specific framework – The Globus Toolkit [59]. The WS-Interoperability implementation, on the other hand, presents a more general and interoperable interface using the guidelines from the WS-I extension. Both implementations expose services that map operations directly to the functional resources of the core. A service coupled with a resource is generally referred to as WS-Resource.

Resources, in general, not only have a benefit of holding a state and having a life cycle, but they can also expose other management properties, such as auditing and accounting. They can be created dynamically, and provide other desirable effects through their design model. An example of such effects is the separation of functional components. The business logic and the presentation can thus be implemented separately from each other. The specification of OGSA-DAI resources can be seen in [60]. A WS-I type implementation provides an easily consumable service, while keeping the application WSRF-compliant. This work concerns with the implementation of a WS-I type presentation layer.

In total there are six service resources in OGSA-DAI and, consequently, also six types of services representing these resources in the presentation layer. In order for the implementation

to work, the following services have to be implemented and mapped to their corresponding resources:

- Data request execution service represents the data request execution resource for clients to submit requests. It encases the workflow execution engine, session management and task monitoring functions;
- Data resource information service that can be used by clients to query information about a data resource, e.g. product name, vendor, version and schema. It represents the data resource information resource;
- Data sink service that is typically used in federation services. Underlying data services can push data to the temporary data sink storage of the federation service, where further processing and merging of data takes place;
- Data source service to extract data from a source;
- Service for managing sessions;
- Request management service, which is primarily used for asynchronous requests, in particular, to extract the status of the asynchronous query execution and its data.

The data request execution service (DRES) and its corresponding resource (DRER) offer the primary interaction point for the client. This service passes all client requests to the resource and starts the execution. The following work description focuses solely on this service. Other services are less critical for the primary use cases, and are structured similarly to the DRES. Thus, they will not be described in further detail.

Following the specification of the services and their mappings to functional components, the operations, parameters and the types of information exchanged between the layers have to be described. Since the presentation layer relays most of the parameters from the client request to the service resource, the following two lists of information exchange, as described by [61]. It serves as a guide to complex data types for service calls.

Information passed from presentation layer to business logic layer

The presentation layer usually forwards information extracted from SOAP messages and derives its data types. These data types are then used in the business logic layer to execute the process. Following information is relayed to the business logic layer:

- Client proxy certifies the user in a web independent format;
- Receive data;

- Perform documents (sequences of tasks) from clients;
- DAI-Core configuration information, including data resource drivers, data resource URIs, database user names and passwords, information on supported activities and the legal form of Perform documents.

Information passed from business logic layer to presentation layer

The business logic layer constructs corresponding response types and forwards it to the presentation layer for SOAP-message wrapping. Following information is relayed back to the presentation layer:

- Response documents;
- Data for delivery;
- Data resource schema;
- Information on data resource-related activities that can be requested by the user within perform documents.

Since WSRF works in conjunction with several other WS-* extensions, these need to be considered as well. As pointed out in [62]:

> The WSRF extension to WS-Addressing - the notion of WS-resource qualified endpoint references - is used to identify to OGSA-DAI web services the resource at which a specific operation is targeted. For functionality relating to lifetime management and the access of resource properties OGSA-DAI web services implement operations conformant with the WS-ResourceProperties and WS-ResourceLifetime specifications.

A total of three presentation layer-relevant parts are needed to implement an OGSA-DAI service:

- the service port types - the web service interfaces, that group operations related to the resource operations;
- the service implementation - concrete implementations of a service port type;
- data types - realization of the information that that will be exchanged by services.

The specifications for these parts are defined by the OGSA-DAI team in WSDL contracts. This ensures OGSA-DAI's interoperability between different presentation layers, and also provides a better maintenance and development of the these services. WSDLs related to OGSA-DAI are described in Section 6.4, while the knowledge gathered in this chapter can be used to provide a component diagram of OGSA-DAI.

5.2 OGSA-DAI component diagram

The following component diagram in Figure 5.2 depicts how resources and other interfaces fit and combine components for constructing an OGSA-DAI service instance.

The general components include the core, the extensions and the presentation. The presentation layer contains the server endpoint that includes the port types, the implementation and the data types needed for communication. It exposes the web service interface to the client and implements the communication facilities, such as message marshaling and unmarshaling, XML parsing and other message processing mechanisms. As mentioned before, services are mapped to the core functionality of OGSA-DAI through service resources.

The OGSA-DAI core contains the workflow execution engine that implements the management and execution of workflows and its corresponding activities. The core holds classes that are common for both client and server, classes for resource and session management and error handling. The setup of the OGSA-DAI middleware is managed through the configuration loader and OGSA-DAI context components.

The extensions provide functionality and connectivity for data sources. They are also represented via resources to the presentation layer, generally referred to as data resources. These wrappers implement activities that execute core functionalities of OGSA-DAI in relation to their exposing data source, thus integrating them into the framework. Some activities implement their own connection drivers, such as RDF with Apache Jena, while the relational resource uses the data component of OGSA-DAI that contains JDBC drivers out-of-the-box.

By considering architectural components and structure of OGSA-DAI, the sequence of tasks needed to carry out the primary use cases can be presented.

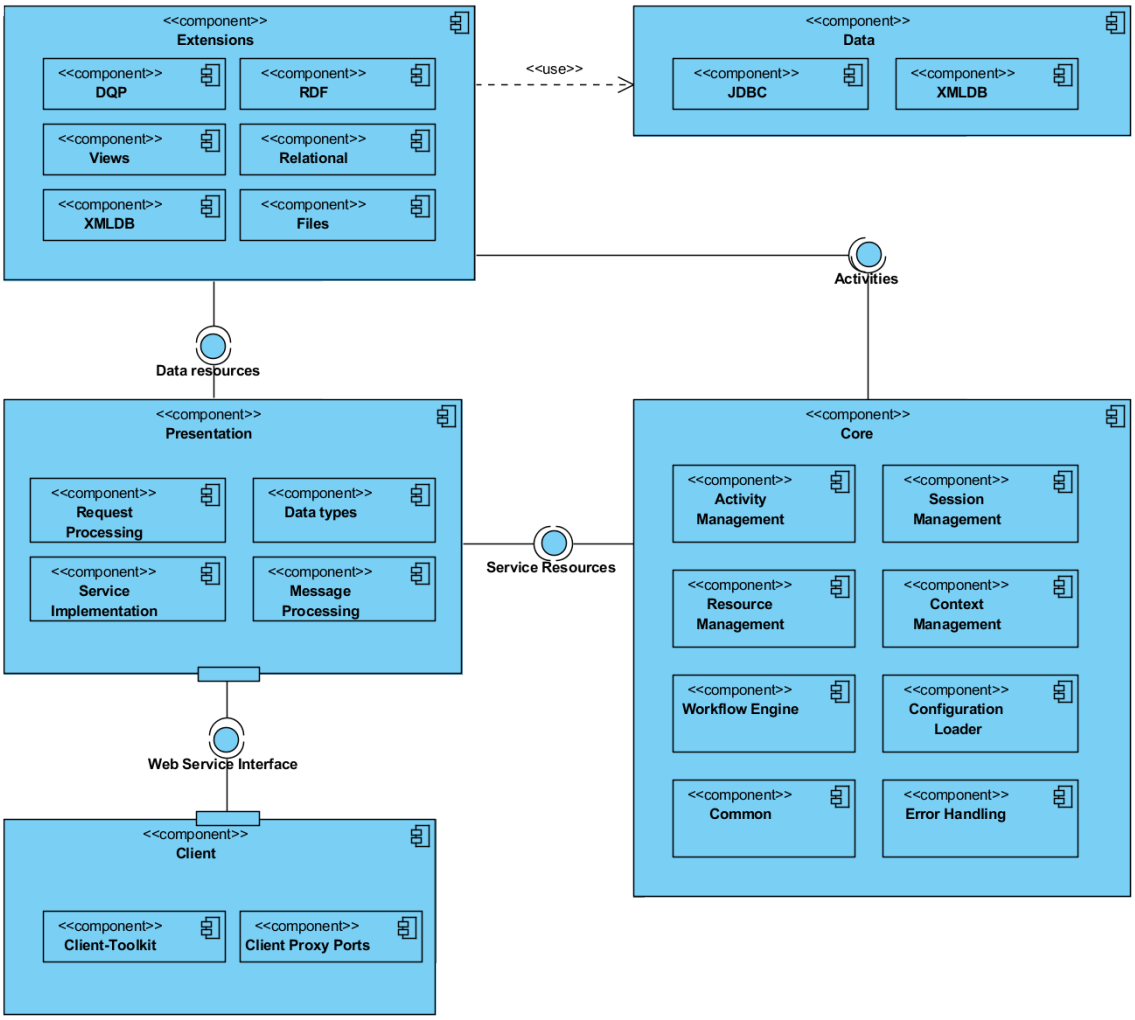


Figure 5.2: This component diagram provides an overview of the data services and its components based on the OGSA-DAI framework. The extensions and core components are represented as resources in the presentation layer. The client addresses them when sending a request.

5.3 Querying a data service

Figure 5.3 depicts the handling of synchronous requests by the data service. Calls (1) and (2) show that the middleware context and service resources are initialized prior to the request.

The call begins with the client app, which formulates a query and specifies parameters for the DRES. The client then uses the OGSA-DAI Client-Toolkit classes (3) to create activities and compose a workflow (4). The client proxy is then created and invoked (5,6). The proxy is responsible for constructing the request SOAP message (6.1) out of the workflow specified by the user, and for sending the message via the network (6.2). The message is directed to the DRES of the specified instance server. The DRES, being an interface implementation, delegates the request to a pluggable provider, in this case the resource provider (6.2.2) The resource provider processes the SOAP message, requests the DRER instance from the OGSA-DAI context and delegates the server-side perform document to it (6.2.2.2). After that, the DRER starts the workflow execution engine (not shown in the diagram) that is responsible for executing the activities defined by the client-workflow. Activities are executed in threads, while data is moved through pipes. The alt-segment models the interactions of the SQLActivity (6.2.2.3.2), since this activity queries the actual SQL data source. The DRER gets a data resource instance from the OGSA-DAI context that uses JDBC drivers to connect and query SQL server. Data is transfered through pipes as soon as they become available. Upon completion of all tasks, the DRER returns the execution result to the Client (6.2.2.3.4).

5.4 Querying a federation service

While Figure 5.3 shows requests for a data service, Figure 5.4 depicts how a federation service communicates with multiple data services. Since the interface of the DRES is generic to both types of services, the federation service uses the same client toolkit classes as the client to communicate (2.1-2.4) with the underlying data services. The federation service uses the OGSA-DQP extension, enabled within the federation service, which constructs a query plan for each data service requested in the federated query (2.3). Prior to that, however, it will need to extract the schemas (2.2) of the requested data services. After the evaluation components of the federation service have completed building the query plan, the querying of each separate data service is accomplished through the client toolkit, as shown in figure 5.3. While the client still uses the DRES to send requests, data services communicate with and respond to the Data Sink Service of the data federation service to process data and return it to the client. The federation service stores the data temporarily in case further processing is needed.

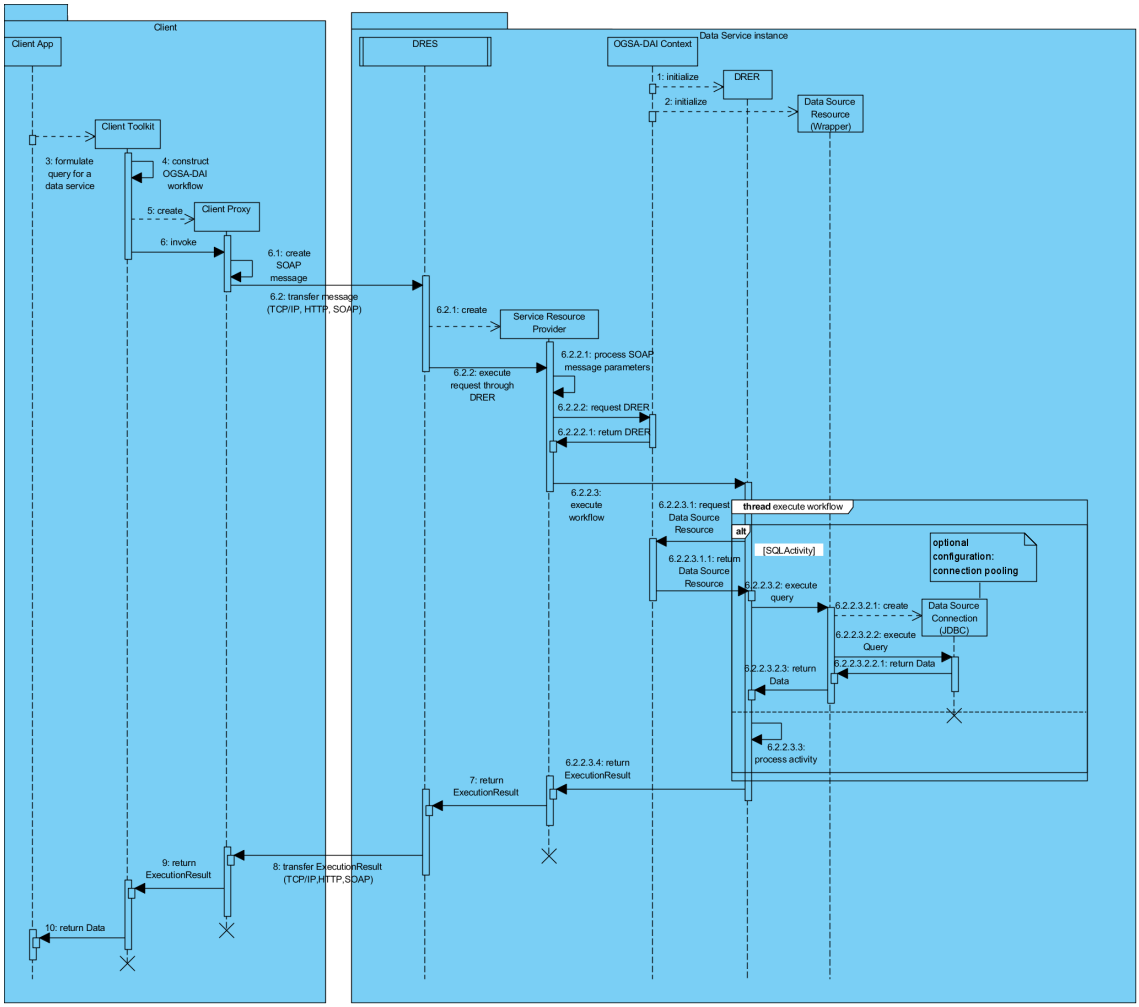


Figure 5.3: This diagram underlines the communication procedure of a client and a data service, involving the service resources that are needed to process a specific client request.

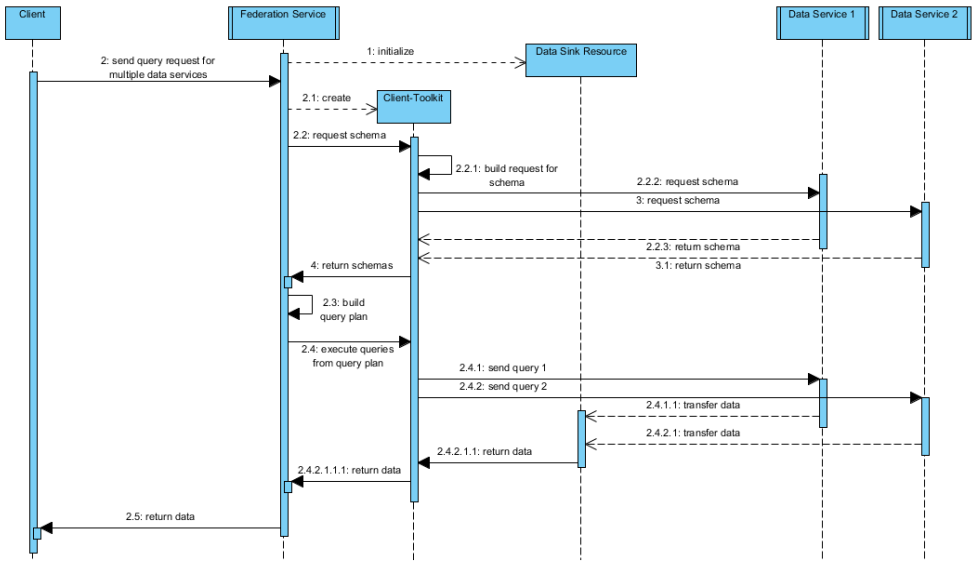


Figure 5.4: The above sequence describes the communication procedure used by a federation service for querying of multiple data services via OGSA-DQP.

5.5 Implementation modeling

The following class diagram serves as the implementation guide. Several components, such as error handling, security delegation and implementations classes for interfaces that relate to OGSA-DAI and Apache CXF are omitted for simplicity (see Apache CXF common and OGSA-DAI common packages). The diagram models classes significant to the implementation of the presentation layer. Since more that one class uses these implementations (especially Apache CXF related), the references to them were generalized as references to the whole package. Functions and class variables are meant provide a general overview of the functionality of the class (still incomplete).

The implemented classes can be the server side, or client side. The proxy implementation package consists mainly of classes that implement the service from a WSDL document specification. These classes can be generated automatically using the WSDL2Java included in the Apache CXF release. They contain data types for information exchange, and, importantly, the web service interface *WSDataRequestExecutionPortType*. As the modeling focuses on the DRES interface, other service interfaces are omitted. Although other services need to be implemented, they can relate to this model in the same manner as the DRES interface does. Similar to the Apache Axis presentation layer classes, each class implemented for the new presentation layer is denoted with the 'CXF' prefix. The functional implementation is a direct port of the functionality of the old presentation layer. Changes will be kept to a minimum. The main difference in the coding lies in the communication and message processing parts of the implementation.

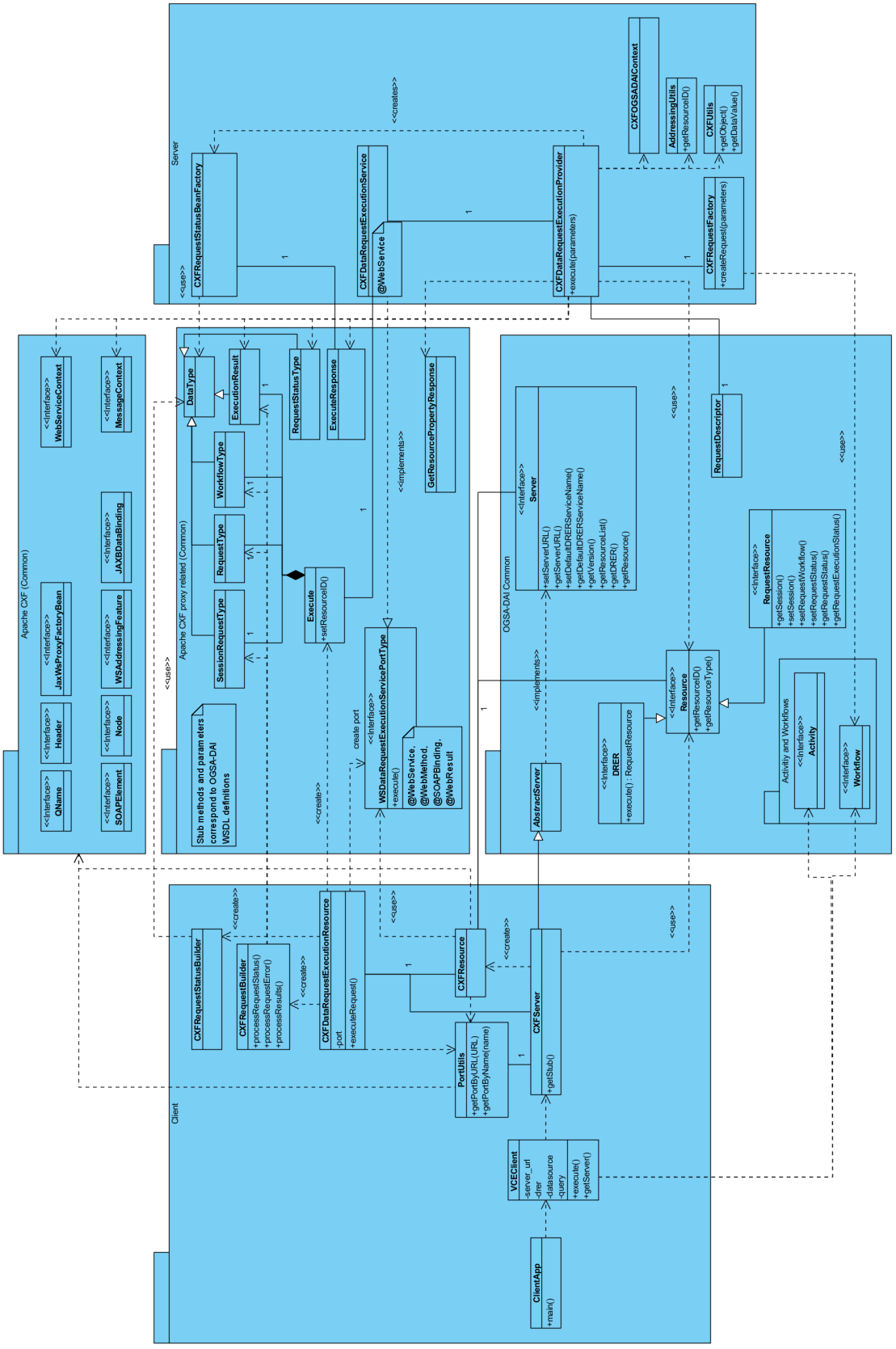


Figure 5.5: This class diagram describes the relations between the presentation layer specific classes denoted by the prefix 'CXF' and the core functional classes of OGSA-DAI and Apache CXF. In addition to Apache CXF and OGSA-DAI libraries, the server and client-side classes make use of the service proxies, generated by Apache CXF with respect to the WSDL.

The client package contains the *ClientApp* class, with the starting *main*-method that creates and invokes the *VCEClient* client instance. This class constructs the activities and workflow, including setting other configuration parameters to produce a web service call. It uses the *CXFServer*, which is the client-side representation of the server. *CXFServer* mostly implements methods for resolving URL and providing the WSDL contract. It uses other classes indirectly to invoke the actual call. *CXFServer* works with the *PortUtils* class, that resolves the different port types for the different services of an OGSA-DAI instance. The class uses the *JaxWsProxyFactoryBean*, along with other CXF-related classes, to instantiate the web service proxy bean for the client. *PortUtils* is the only new class added to the preexisting Axis*-type classes. It provides static methods to resolve CXF ports via URL or resource name. *CXFServer* also creates and uses *CXFResource* and *CXFDataRequestExecutionResource* classes that represent the corresponding resources in OGSA-DAI. The *CXFDataRequestExecutionResource* it the class that actually calls the web service method. The *CXFRequestBuilder* and *CXFRequestStatusBuilder* classes mainly help the WS-Resource representation class *CXFDataRequestExecutionResource* to construct the request document through the proxy-related data type classes.

On the server-side, the *CXFDataRequestExectionService* is the implementation of the web service interface *WSDataRequestExecutionPortType*. The class has no real functionality as it only passes the requests onto the corresponding functionality providers. In this case, the *CXFDataRequestExecutionProvider*. The provider resolves the resource id through the *AddressingUtils*, providing mechanisms for WS-Adressing, and requests a DRER instance from global *OGSADAIContext*. After that the *CXFRequestFactory* is given the request parameters to construct the workflow perform document. Properties taken from the *MessageContext*, that is a interface for the SOAP message, and inserted into this document, after which DRER is given the task to execute it. *CXFRequestStatusBeanFactory* is then used to create a *ExecuteResponse* and return it to the client.

5.5.1 VCE Integration

The solution adopted for integrating data services into the VCE is show in Figure 5.6. The interface specifies methods for selecting a workflow, formulating a query and saving results on the local storage medium. The *WorkflowBuilder* class constructs workflows of SPARQL or SQL query types, and deliver data in the CSV or the XML format. Two separate client implementations will allow two methods of communications following REST or SOAP patterns. The VCE client uses property files to save configurations and execution parameters. These properties and their values are defined in *VCEClientConstants*. A property configuration can thus be given to the *VCEClientFactory* to provide the client with preconfigured settings.

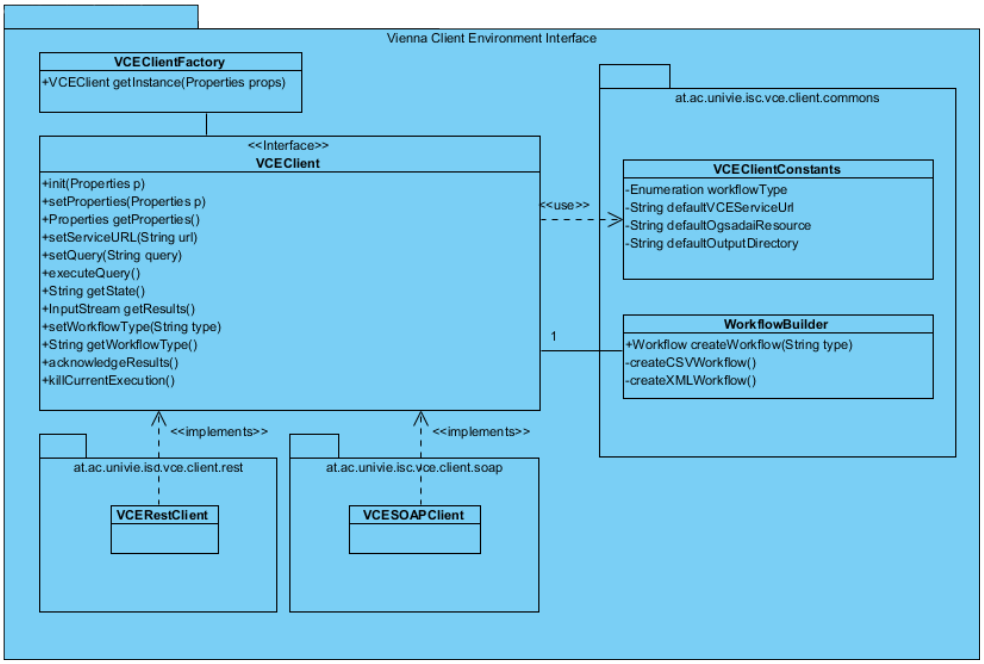


Figure 5.6: Class diagram for the VCE Client Interface. This interface is used to integrate data services into the VCE.

The service deployment tool, a separate utility for data services, mainly consists of Apache Ant Tasks to accomplish automated deployment of the service. As mentioned earlier, the software is planned to be preconfigured on an virtual image and thus delivered as a virtual appliance. The image includes all the necessary software, configurations and libraries for a convenient deployment of data services. When executing the deployment tool, it will create and deploy the web service application into Apache Tomcat's container. The tool is related to the administrative use-case of this application.

5.6 Summary

This chapter discussed and defined the middleware and web service framework that will be used to implement the data access and integration services. The diagram section provides a guide for the implementation by analyzing the process of communication of the services, including components and relevant classes.

The next chapter will discuss the implementation details with specific configuration documents, source code and issues relating to the implementation.

Chapter 6

Implementation

Nothing resolves design issues like an implementation.
– J. D. Horton

This chapter discusses the implementation of the OGSA-DAI presentation layer with Apache CXF web service framework and its integration in the VCE. The web service definitions, communication messages, web service interface, including their implementation and configuration, are described. The source code is provided in abbreviated form, highlighting the code lines critical to the implementation. Issues regarding the system development and their solutions are discussed at the end of this chapter.

6.1 Preamble

The OGSA-DAI's development team begun working on their own implementation of OGSA-DAI with CXF in June 2010 [63]. This work apparently came to a halt in January 2011, with issues regarding the implementation [64]. In December. 2011 OGSA-DAI 4.2 based on Jersey Technology was released. It implemented a RESTful presentation layer for OGSA-DAI. The specifications of the VPH-Share Project, however, require a SOAP-based web service presentation layer. Annotated OGSA-DAI source code can be found in [65].

6.2 The current state of the OGSA-DAI-native CXF layer

In principle it is possible to leave the functional code of OGSA-DAI's presentation layer as-is. Changes include the customization and mapping of data objects to the JAXB standard and the

generation and customization of the service proxies.

In his blog (see, [63]), an OGSA-DAI developer states that several data types had to be adjusted to accommodate the JAXB standard, mentioning the data type xs:dateTime being changed from XMLGregorianCalendar to *java.util.Calendar* as an example. Furthermore, interfaces and proxies are generated automatically using the tools provided by Apache CXF. In particular, the wsdl2java tool can be used to generate Java interface and stub classes given the contract WSDL documents. In Apache CXF, however, the generated proxies utilize lists, as opposed arrays utilized by Apache Axis. Several changes had to be made to resolve this particular issue. Further revisions include some adjustments of Request management and Session Management Services, WS-ResourceLifetime and Request Builder operations, execution of workflows and fault types. In addition, several utility classes such as *AddressingUtils* are implemented to allow extraction of message objects and headers.

The most recent revisions to the source code include adjustments to the service front-end interfaces. Concerning future developments, [63] states that a discussion about implementing a RESTful layer for OGSA-DAI has begun, while the CXF presentation layer implementation for OGSA-DAI has been abandoned.

6.3 WSDL definitions

The implementation details start with the definition of the web services. As described in Section 5.1, OGSA-DAI provides a set of WSDL documents for specifying web services. The modular structure of the OGSA-DAI WSDL schema definitions is shown in Appendix A.

Listing 6.1 show the binding definitions for the DRES. The WSDL document of a particular service defines the service name, its port and SOAP protocol bindings. The import tag denotes that another document is imported to complete this WSDL, see the execution bindings definition for the *DataRequestExecutionService*.

Listing 6.1: data_request_execution_service_service.wsdl

```
1  <wsdl:definitions name="DataRequestExecutionService" targetNamespace="
       http://ogsadai.org.uk/namespaces/2007/04/service/execution/service"
       xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:binding="
       http://ogsadai.org.uk/namespaces/2007/04/service/execution/bindings"
       xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
2  <wsdl:import namespace="http://ogsadai.org.uk/namespaces/2007/04/service/
       execution/bindings" location="data_request_execution_service_bindings.
       wsdl"/>
3  <wsdl:service name="DataRequestExecutionService">
4    <wsdl:port name="WSDataRequestExecutionServicePortTypePort" binding="
         binding:WSDataRequestExecutionServicePortTypeSOAPBinding">
```

```
5       <soap:address location="http://localhost:8080/services/
            DataRequestExecutionService"/>
6     </wsdl:port>
7   </wsdl:service>
8 </wsdl:definitions>
```

Listing 6.2 describes operation bindings for the service. Here, each operation is defined for the client to execute, including method input, output and fault parameters. The *types* section defines the service properties type and maps it to WS-Addressing schema. The document includes several other types of generic definitions that construct the complex data types for requests and responds; these will be omitted here. Further note that namespaces are similar to Java class packaging and is mainly used for classification, structuring and identification of web services elements.

Listing 6.2: data_request_execution_service_port.wsdl

```
1  <wsdl:definitions name="DataRequestExecutionService" targetNamespace="http:
       //ogsadai.org.uk/namespaces/2007/04/service/execution" xmlns:...>
2    <wsdl:import namespace="http://ogsadai.org.uk/namespaces/2007/04/service/
         execution" location="../../generic-wsdl/
         data_request_execution_port_type.wsdl"/>
3    <wsdl:import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
         ResourceProperties-1.2-draft-01.wsdl" location="../../../wsrf/
         properties/WS-ResourceProperties.wsdl"/>
4    <wsdl:import namespace="http://ogsadai.org.uk/namespaces/2007/04/service/
         intrinsics" location="../../generic-wsdl/intrinsics_port_type.wsdl"/>
5    <wsdl:import namespace="http://ogsadai.org.uk/namespaces/2007/04/service/
         resolver" location="../../generic-wsdl/ws_epr_resolver_port_type.wsdl"
         />
6    <wsdl:types>
7      ...
8    </wsdl:types>
9
10   <wsdl:portType name="WSDataRequestExecutionServicePortType"
         wsrp:ResourceProperties="
         execution:DataRequestExecutionResourceProperties">
11     <wsdl:operation name="execute">
12       <wsdl:input message="execution:ExecuteInputMessage"/>
13       <wsdl:output message="execution:ExecuteOutputMessage"/>
14       <wsdl:fault name="ServerFault" message="execution:ServerFault"/>
15       <wsdl:fault name="ResourceUnknownFault" message="
             execution:ResourceUnknownFault"/>
16       <wsdl:fault name="ClientFault" message="execution:ClientFault"/>
17     </wsdl:operation>
18     ...
19   </wsdl:portType>
20 </wsdl:definitions>
```

The binding WSDL in Listing 6.3 for the DRES provides specifications for the communi-

cation with SOAP messages. The *binding* instructs the use of SOAP messages through HTTP for communication. The messages of the operations are provided in the Document/literal-style, which encodes arguments and return values in a certain way. WSDL message parts are references to elements defined in the XML Schema. While this style is WS-Interface complaint and the documents are easily validated, it leads to a more complicated WSDL because of further definition of data types. Further information of the encoding styles in WSDL can be viewed in [66].

Listing 6.3: data_request_execution_service_bindings.wsdl

```
<wsdl:definitions name="DataRequestExecutionService" xmlns:...>
  <wsdl:import namespace="http://ogsadai.org.uk/namespaces/2007/04/service/
      execution" location="data_request_execution_service_port_type.wsdl"/>
  <wsdl:binding name="WSDataRequestExecutionServicePortTypeSOAPBinding"
      type="porttype:WSDataRequestExecutionServicePortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/
        soap/http"/>
    <wsdl:operation name="execute">
      <soap:operation soapAction="http://ogsadai.org.uk/namespaces/2007/04/
          service/execution/WSDataRequestExecutionServicePortType/
          executeRequest"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="ServerFault">
        <soap:fault name="ServerFault" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="ResourceUnknownFault">
        <soap:fault name="ResourceUnknownFault" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="ClientFault">
        <soap:fault name="ClientFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
    ...
  </wsdl:binding>
</wsdl:definitions>
```

Further included to complete service definitions are WSDL document parts that define the specific data and message types. The final definition reviewed here is the request message type definition, see Listing 6.4. As can be seen in the code provided below, the message consists of workflow and activity definitions and their associated types. Each activity has an input and output type mapped to InputStreamType and OutputStreamType. Other parameters of activities are defined in the *attribute* directive. Activity name and class instance are required because the input and output of one activity can be connected to another. If the activity processes a certain

data resource, it will be assigned a resource id.

Listing 6.4: Request.xsd

```
<xsd:schema targetNamespace="http://ogsadai.org.uk/namespaces/2007/04/
    types"
xmlns:tns="http://ogsadai.org.uk/namespaces/2007/04/types"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">

<!-- Root element -->

<xsd:element name="request" type="tns:RequestType"/>

<xsd:complexType name="RequestType">
  <xsd:sequence>
    <xsd:element name="workflow" type="tns:WorkflowType"/>
  </xsd:sequence>
</xsd:complexType>

<!-- Elements describing a workflow -->
<xsd:complexType name="WorkflowType">
  <xsd:choice>
    <xsd:element name="parallel" type="tns:ParallelType"/>
    <xsd:element name="sequence" type="tns:SequenceType"/>
    <xsd:element name="pipeline" type="tns:PipelineType"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="SequenceType">
  <xsd:sequence>
    <xsd:element name="workflow" type="tns:WorkflowType" maxOccurs="
        unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="PipelineType">...</xsd:complexType>
<xsd:complexType name="ParallelType">...</xsd:complexType>

<!-- Activity type -->
<xsd:complexType name="ActivityType">
  <xsd:sequence>
    <xsd:element name="inputs" type="tns:InputsType"/>
    <xsd:element name="outputs" type="tns:OutputsType"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="instanceName" type="xsd:ID" use="required"/>
  <xsd:attribute name="resource" type="xsd:string" use="optional"/>
</xsd:complexType>

<!-- Activity inputs type -->
<xsd:complexType name="InputsType">
  <xsd:sequence>
```

```
47      <xsd:element name="input" minOccurs="0" maxOccurs="unbounded">
48        <xsd:complexType>
49          <xsd:choice>
50            <xsd:element name="inputStream" type="tns:InputStreamType"/>
51            <xsd:element name="inputLiteral" type="tns:DataType"
52                         maxOccurs="unbounded"/>
53          </xsd:choice>
54          <xsd:attribute name="name" type="xsd:string" use="required"/>
55        </xsd:complexType>
56      </xsd:element>
57    </xsd:sequence>
58  </xsd:complexType>
59
60  <!-- Activity outputs type -->
61  <xsd:complexType name="OutputsType">
62    <xsd:sequence>
63      <xsd:element name="outputStream" type="tns:OutputStreamType"
            minOccurs="0" maxOccurs="unbounded"/>
64    </xsd:sequence>
65  </xsd:complexType>
66
67  <!-- Activity input stream type -->
68  <xsd:complexType name="InputStreamType">
69    <xsd:attribute name="pipe" type="xsd:IDREF" use="required"/>
70  </xsd:complexType>
71
72  <!-- Activity output stream type -->
73  <xsd:complexType name="OutputStreamType">
74    <xsd:attribute name="name" type="xsd:string" use="required"/>
75    <xsd:attribute name="pipe" type="xsd:ID" use="required"/>
76  </xsd:complexType>
77
78  <!-- Data values -->
79  <xsd:complexType name="DataType">
80    <xsd:choice>
81      <xsd:element name="string" type="xsd:string"/>
82      <xsd:element name="charArray" type="xsd:string"/>
83      <xsd:element name="binary" type="xsd:base64Binary"/>
84      <xsd:element name="float" type="xsd:float"/>
85      <xsd:element name="double" type="xsd:double"/>
86      <xsd:element name="int" type="xsd:int"/>
87      <xsd:element name="long" type="xsd:long"/>
88      <xsd:element name="boolean" type="xsd:boolean"/>
89      <xsd:element name="date" type="xsd:dateTime"/>
90      <xsd:element name="listBegin" type="tns:ListBeginType"/>
91      <xsd:element name="listEnd" type="tns:ListEndType"/>
92    </xsd:choice>
93  </xsd:complexType>
94
95 </xsd:schema>
```

To summarize the request definition provided above, it consists of a specific workflow containing one or more activities that have a name, a generated instance name and an optional resource id. An activity transmits either streamed data or literals (strings). Input and output streams are data pipes that can transport several data types listed at the end of the document. These can be of different data value types, including primitive and complex data structures. The *xsd:base64Binary* data type denotes the use of the MTOM extension for binary data transfer for a particular parameter. This functionality is provided by Apache CXF.

How the activities are connected will explained in Section 6.9 below.

6.4 SOAP communication

Applying the strict definition of the data services produces similar SOAP messages, regardless of the presentation layer type or framework that implements it.

An example of a request SOAP message for fetching data of a SQL data source may look in the following way (Listing 6.5):

Listing 6.5: SOAP request message

```
<soap:Header>
  <ResourceID xmlns="http://ogsadai.org.uk">DataRequestExecutionResource</
      ResourceID>

  <!-- WS-Addressing directives -->
  <Action>...</Action><MessageID>...</MessageID><To>...</To><ReplyTo>...</
      ReplyTo>
</soap:Header>

<ns2:execute xmlns="...">
  <ns4:request>
    <ns3:workflow>
      <ns3:pipeline>

        <ns3:activity name="SQLQuery" resource="CRIM1">
          <ns3:inputs>
            <ns3:input name="expression"><ns3:string>SELECT * FROM eventid;
                </ns3:string></ns3:input>
          </ns3:inputs>
          <ns3:outputs><ns3:outputStream name="data" pipe="pipeID1"/></
              ns3:outputs>
        </ns3:activity>

        <ns3:activity name="TupleToWebRowSetCharArrays">
          <ns3:inputs><ns3:input name="data"><ns3:inputStream pipe="pipeID1
              "/></ns3:input></ns3:inputs>
```

```
22          <ns3:outputs><ns3:outputStream name="result" pipe="pipeID2"/></
                ns3:outputs>
23        </ns3:activity>
24
25        <ns3:activity name=DeliverToRequestStatus">
26          <ns3:inputs><ns3:input name="input"><ns3:inputStream pipe="
                pipeID2"/></ns3:input></ns3:inputs>
27          <ns3:outputs/>
28        </ns3:activity>
29
30      </ns3:pipeline>
31    </ns3:workflow>
32  </ns4:request>
33 <ns2:isSynchronous>true</ns2:isSynchronous>
34 </ns2:execute>
```

The header contains the id of the addressed service resource (*DataRequestExecutionResource*), and a mapping through WS-Addressing directives.

The body specifies a call the execute operation that consists of a pipeline workflow. The workflow consists of three activies and their respective parameters: data querying (*SQLQuery* activity), data transformation (*TupleToWebRowSetCharArrays*) and delivery (*DeliverToRequestStatus*). The delivery activity specifies that data should be sent through the *RequestStatus*-object, so they can be viewed in response message of this request. Activities are connected through input and output pipes. Since the first activity has no input and the last no output, these nested parameters are missing.

The response specific to this request, see Listing 6.6, contains all relevant information about the activities, the status of the request, schema and the data:

Listing 6.6: SOAP response message

```
1  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
2    <soap:Body>
3      <ns3:executeResponse xmlns="...">
4        <ns2:requestStatus>
5          <requestDetails status="COMPLETED" id="..."/>
6          <activity status="COMPLETED" instanceName="uk.org.ogsadai.
               DeliverToRequestStatus"/>
7          <activity status="COMPLETED" instanceName="uk.org.ogsadai.
               TupleToWebRowSetCharArrays"/>
8          <activity status="COMPLETED" instanceName="uk.org.ogsadai.SQLQuery"
               />
9
10         <result resultName="result" activityInstanceName="uk.org.ogsadai.
               DeliverToRequestStatus">
11         <data><charArray>
12           <webRowSet xmlns="...">
13             <properties>
```

```
14                <command>null</command>
15                <concurrency></concurrency>
16                <!-- ...further properties //-->
17              </properties>
18              <metadata>
19                <column-count>4</column-count>
20                <column-definition>
21                <column-index>1</column-index>
22                <auto-increment></auto-increment>
23              <!-- ...further metadata //-->
24              </metadata>
25
26            <!-- actual data //-->
27            <data><charArray><currentRow>
28              <columnValue>10001</columnValue>
29              <columnValue>10004</columnValue>
30              <columnValue>0</columnValue>
31              <columnValue>0</columnValue>
32            </currentRow></charArray></data>
33            <data><charArray>...<currentRow></data>
34             ...
35          </result>
36        </ns2:requestStatus>
37      </ns3:executeResponse>
38    </soap:Body>
39  </soap:Envelope>
```

6.5 Proxy classes

Classes that derive from the WSDL comprise endpoint-interface, implementation and data type. These classes represent the service proxies for the application. Since the contract WSDL document is provided by OGSA-DAI and is generic throughout any presentation layer, the wsdl2java is used for generating the CXF-specific proxies in a contract-first approach for building the service. The tool translates the documents definition directly into JAX-WS-based web services beans that are injected into the Apache CXF framework.

Once provided with the OGSA-DAI WSDL schema, Apache CXE generates the following endpoint-interface, presented in Listing 6.7. The interface defines the web service with its SOAP binding(@*WebService* and @*SOAPBinding*) and annotates the methods, together with their corresponding result types (@*WebMethod* and @*WebResult*) and exception faults.

Listing 6.7: WSDataRequestExecutionServicePortType.java

```
@WebService(targetNamespace = "http://ogsadai.org.uk/namespaces/2007/04/
    service/execution", name = "WSDataRequestExecutionServicePortType")
@SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
public interface WSDataRequestExecutionServicePortType {

    @WebResult(name = "executeResponse", targetNamespace = "http://ogsadai.
        org.uk/namespaces/2007/04/service/execution", partName = "parameters
        ")
    @WebMethod(action = "http://ogsadai.org.uk/namespaces/2007/04/service/
        execution/WSDataRequestExecutionServicePortType/executeRequest")
    public ExecuteResponse execute(
        @WebParam(partName = "parameters", name = "execute",
            targetNamespace = "http://ogsadai.org.uk/namespaces/2007/04/
            service/execution")
        Execute parameters
    ) throws ServerFault, ClientFault, ResourceUnknownFault;

    //other web service methods
    ...
}
```

The parameter message types are generated as data and message beans related to the data type definition in WSDL. The port type makes the use of several complex data types. The input *Execute* and the output *RequestStatusType* types of the *execute* method can be seen below.

Listing 6.8: Execute.java and RequestStatusType.java

```
//---Execute.java---

@XmlType(name = "", propOrder = {"request","requestID","session","
    isSynchronous"})
@XmlRootElement(name = "execute")
public class Execute
    implements Serializable {

    @XmlElement(namespace = "http://ogsadai.org.uk/namespaces/2007/04/types
        ")
    protected RequestType request;
    protected String requestID;
    protected SessionRequestType session;
    protected Boolean isSynchronous;

    //below follow getter and setter for the above variables
    ...
}

//---RequestStatusType.java---

@XmlAccessorType(XmlAccessType.FIELD)
```

```
@XmlType(name = "", propOrder = {"requestStatus","requestID","sessionID"})
@XmlRootElement(name = "executeResponse")
public class ExecuteResponse implements Serializable {
    @XmlElement(namespace = "http://ogsadai.org.uk/namespaces/2007/04/types
        ")
    protected RequestStatusType requestStatus;
    protected String requestID;
    protected String sessionID;

    //below follow getter and setter for the above variables
    ...
}
```

6.6 Server implementation

Although the wsdl2Java tool generates service implementations and provides executable classes, these will not be used because doing so requires rewriting the code provided by the tool. Thus, they are developed manually in accordance with the OGSA-DAI's coding rules. The following code in Listing 6.9 shows a service implementation of the above interface specification without exception handling, which are omitted for brevity.

Listing 6.9: CXFDataRequestExecutionService.java

```
@javax.jws.WebService(
        serviceName = "DataRequestExecutionService",
        portName = "WSDataRequestExecutionServicePortTypePort",
        targetNamespace = "http://ogsadai.org.uk/namespaces/2007/04/service
            /execution/service",
        endpointInterface = "uk.org.ogsadai.service.cxf.execution.
            WSDataRequestExecutionServicePortType")
public class CXFDataRequestExecutionService
        implements WSDataRequestExecutionServicePortType {
    @Resource
    private static WebServiceContext wsContext;
    public void init(Object context){...}
    public void destroy(){...}

    public ExecuteResponse execute(Execute parameters)
        throws ServerFault, ResourceUnknownFault, ClientFault {
        CXFDataRequestExecutionProvider provider =
                new CXFDataRequestExecutionProvider();

        return provider.execute(parameters);
    }
    ...
}
```

The web service definition annotation defines specific properties of the web service, including the name, the port and the endpoint interface. The *WebServiceContext* can be accessed through this class. It provides mechanisms for custom message processing needed later. The `@Resource` reference marks the `WebServiceContext` as a dependency for the current class. Since the context is being managed by the CXF framework, it is injected at runtime following the inversion-of-control principle. If necessary, the *init*-function takes care of OGSA-DAI context initialization. As can be deducted from the source code, the implementation of the *execute* method only passes the request parameters to the *CXFDataRequestExecutionProvider*, see Listing 6.10. It is the provider class, however, that actually processes the requests.

Listing 6.10: CXFDataRequestExecutionProvider.java

```
1  public class CXFDataRequestExecutionProvider {
2      public ExecuteResponse execute(Execute parameters)
3          throws ServerFault, ResourceUnknownFault, ClientFault {
4          DRER drer = null;
5          try {
6              ResourceID id = AddressingUtils.getResourceID();
7              Resource resource = OGSADAIContext.getInstance().
8                  getResourceManager().getPublicResource(id);
9              if (resource instanceof DRER) { drer = (DRER) resource; }
10             else { throw new ResourceUnknownException(id); }
11         } catch (Exception e) {...}
12
13         // Build DRER request descriptor.
14         String requestIDStr = parameters.getRequestID();
15
16         ResourceID requestID = null;
17         if ((requestIDStr != null) && !("".equals(requestIDStr.trim()))) {
18             requestID = new ResourceID(requestIDStr);
19         }
20
21         // Session-handling
22
23         // Get request bean and convert to workflow.
24         Workflow workflow = null;
25         try
26         {
27             CXFRequestFactory requestFactory = new CXFRequestFactory();
28             workflow = requestFactory.createRequest(parameters.getRequest()
                   );
29         }
30         catch (Exception e) {...}
31
32         // Construct descriptor.
33         SimpleCandidateRequestDescriptor requestDescriptor =
34                 new SimpleCandidateRequestDescriptor(requestID, sessionID,
35                         createSession, parameters.isIsSynchronous(),
36                         false, workflow);
37
```

```
38          ExecutionResult result = null;
39          try {
40              result = drer.execute(requestDescriptor);
41          } catch (Exception e){...}
42
43          CXFRequestStatusBeanFactory factory = new
                CXFRequestStatusBeanFactory();
44          RequestStatusType requestStatusBean = (RequestStatusType) factory.
45              createBean(result.getRequestStatus());
46          ExecuteResponse response = new ExecuteResponse();
47          response.setRequestID(result.getRequestID().toString());
48          if (result.getSessionID() != null) {
49              response.setSessionID(result.getSessionID().toString());
50          }
51          response.setRequestStatus(requestStatusBean);
52          return response;
53      }
54  }
```

The provider extracts the resource id from the message header and gets the DRER resource instance from the *OGSADAIContext* through the *AddressingUtils* class. After some session-handling, the *CXFRequestFactory* constructs a server-side *workflow* object for the DRER using the web method input parameters. The *workflow* instance is added to the request descriptor, which includes all input parameters for the DRER execution method. The *execute* method of DRER returns the *ExecutionResult* data type, which is passed to *CXFRequestStatusBeanFactory* for generating a *ExecuteResponse*object to be returned to the client.

Other classes in the server implementation include utility classes and handlers. They will not be listed, but explained briefly. The *AddressingUtils* class parses the SOAP message header to extract the resource ID from the message. The *CXFUtils* class converts OGSA-DAI data blocks to CXF-compliant beans and back. It also handles errors thrown by the CXF framework. The *InputHandler* and *OutputHandler* classes are responsible for extracting information from the CXF data beans, managing pipes of the respective activity beans. Similarly, the *WorkflowHandler* manages the CXF-compliant workflow bean.

6.7 Configuration of the service

Since Apache CXF is built upon the Spring Framework, a number of configuration files used by the Spring Framework must be declared. These configuration files are mainly used for context initialization of the web application. The *web.xml* file, see Listing 6.11, describes several deployment elements for the web application. In addition to the default initialization of the Spring Framework context using the *ContextLoaderListener* class, the initialization of OGSA-DAI context class *CXFOGSADAIContextInitializer* is added. As the default *CXFServlet*, the *Data-*

SourceRetrievalServlet is registered for the retrieval of byte and char arrays from an OGSA-DAI data source. Other entries may include declarations of custom mime-types, session configuration, error pages, filter or security contexts.

Listing 6.11: web.xml

```
<web-app>
    <display-name>dai</display-name>
    <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>WEB-INF/beans.xml</param-value>
    </context-param>

    <listener>
      <listener-class>uk.org.ogsadai.service.cxf.context.
          CXFOGSADAIContextInitializer</listener-class>
    </listener>
    <listener>
      <listener-class>org.springframework.web.context.ContextLoaderListener
          </listener-class>
    </listener>

    <servlet>
        <servlet-name>DataSourceRetrievalServlet</servlet-name>
        <display-name>OGSA-DAI Servlet</display-name>
        <servlet-class>uk.org.ogsadai.servlets.DataSourceRetrievalServlet</
            servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>DataSourceRetrievalServlet</servlet-name>
        <url-pattern>/services/DataSourceRetrievalServlet</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>CXFServlet</servlet-name>
        <display-name>CXF Servlet</display-name>
        <description>Apache CXF Endpoint</description>
        <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet
            -class>
        <load-on-startup>100</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>CXFServlet</servlet-name>
        <url-pattern>/services/*</url-pattern>
        <load-on-startup>1</load-on-startup>
    </servlet-mapping>
     ...
</web-app>
```

Next, the declaration of service beans is defined in the *beans.xml* configuration file, see Listing 6.12. The beans represent the services exposed to the network. Each entry defines the endpoints that combine the service interface, its corresponding WSDL definition and the implementation class. Apache CXF uses this file to inject service beans into its engine.

Listing 6.12: beans.xml

```
<beans xmlns=...>

    <import resource="classpath:META-INF/cxf/cxf.xml"/>
    <import resource="classpath:META-INF/cxf/cxf-servlet.xml" />

  <jaxws:endpoint
    xmlns:tns="http://ogsadai.org.uk/namespaces/2007/04/service/execution/
       service"
    id="DataRequestExecutionServiceBean"
    implementor="uk.org.ogsadai.service.cxf.execution.
       CXFDataRequestExecutionService"
    wsdlLocation="schema/ogsadai/cxf-specific-wsdl/dres/
       data_request_execution_service_service.wsdl"
    endpointName="tns:WSDataRequestExecutionServicePortTypePort"
    serviceName="tns:DataRequestExecutionService" address="/
       DataRequestExecutionService">
  </jaxws:endpoint>

  <jaxws:endpoint>...</jaxws:endpoint>
</beans>
```

The configuration examples for the OGSA-DAI data resources are provided in the next section.

6.8 Configuration of a data resource

Configuration of the OGSA-DAI data resources are provided in simple property files. An example is shown in Listing 6.13. Appendix B shows an example of a deployed service directory structure and the location of the files. Consider an example of a simple data resource connecting to a SQL database. Several properties defined in the configuration provide connection parameters to the database server. The resources itself is given a name and a type. The list at the end of the document specifies activities to be executed in relation to the data resource. SPARQL resources are declared in a similar manner, and will not be shown in this document.

Listing 6.13: mysql (resource configuration file)

```
id=mysql
type=uk.org.ogsadai.DATA_RESOURCE
creationTime=null
terminationTime=null
PROPERTIES
END
CONFIG
dai.driver.class=org.gjt.mm.mysql.Driver
dai.data.resource.uri=jdbc:mysql://localhost:3306/database
dai.login.provider=uk.org.ogsadai.LOGIN_PROVIDER
END
ACTIVITIES
uk.org.ogsadai.SQLNestedInClauseQuery=uk.org.ogsadai.SQLNestedInClauseQuery
uk.org.ogsadai.SQLUpdate=uk.org.ogsadai.SQLUpdate
uk.org.ogsadai.SQLParameterisedQuery=uk.org.ogsadai.SQLParameterisedQuery
uk.org.ogsadai.GetAvailableTables=uk.org.ogsadai.GetAvailableTables
uk.org.ogsadai.FilteredSQLQuery=uk.org.ogsadai.FilteredSQLQuery
uk.org.ogsadai.ExtractPhysicalSchemaToXML=uk.org.ogsadai.
    ExtractPhysicalSchemaToXMLSimple
uk.org.ogsadai.ExtractTableSchema=uk.org.ogsadai.ExtractTableSchema
uk.org.ogsadai.SQLStatement=uk.org.ogsadai.SQLStatement
uk.org.ogsadai.SQLBulkLoadTuple=uk.org.ogsadai.SQLBulkLoadTuple
uk.org.ogsadai.SQLQuery=uk.org.ogsadai.SQLQuery
uk.org.ogsadai.SQLParameterisedUpdate=uk.org.ogsadai.SQLParameterisedUpdate
uk.org.ogsadai.SQLNestedInClauseJoin=uk.org.ogsadai.SQLNestedInClauseJoin
END
dataResourceClass=uk.org.ogsadai.resource.dataresource.jdbc.
    JDBCDataResource
```

The DQP and Views resources also have similar configuration files, but they define additional properties and activities. A view resource uses an additional file containing a prepared SQL statement for that particular view. A DQP resource will use an additional configuration file as well to provide mapping to the underlying resources. The mapping may contain the following information:

Listing 6.14: DQPResourceConfig.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<DQPResourceConfig>
    <dataResources>
    <resource url="http://131.130.68.177/vph/CRIM1/services/"
              resourceID="ds"
              alias="CRIM1"
              isLocal="false"/>
    <resource url="http://131.130.68.177/vph/CRIM2/services/"
              resourceID="ds"
         alias="CRIM2"
              isLocal="false" />
    <resource url="http://131.130.68.177/vph/DDS1/services/"
```

```
13                resourceID="ds"
14          alias="DDS1"
15                isLocal="false" />
16     <resource url="http://131.130.68.177/vph/DDS2/services/"
17                resourceID="ds"
18          alias="DDS2"
19                isLocal="false" />
20     </dataResources>
21     <evaluationNodes>
22         <node url="http://131.130.68.177/vph/DQP/services/"
23               isLocal="true" />
24     </evaluationNodes>
25 </DQPResourceConfig>
```

The *DQPResourceConfig* defines the enclosed *dataResources*. In this case, the DQP encapsulates four different data services, none of which is local. There is thus a need for defining the URL for the federation service itself. This information is contained in the *evaluationNodes* tag. This concludes the implementation description for the server.

6.9 Client implementation

The following description of the client implementation starts with a class that utilizes the OGSA-DAI Client-Toolkit functions to construct requests, and invokes the represented service objects to execute them. The class *VCESoapClient*, see Listing 6.15, described in Section 5.5.1, implements the VCE interface *VCEClient*. The source code below implements the *executeQuery* method. Again, logging and error processing features were omitted for description.

Listing 6.15: VCESoapClient.java

```
1  public class VCESoapClient implements VCEClient {
2      private Properties properties;
3      private RequestStatus status = null;
4      private VCEClientState state = VCEClientState.UNINITIALIZED;
5
6      public VCESoapClient(){
7          init(new Properties());
8      }
9
10     public void executeQuery() throws VCEException {
11         String resourceID = properties
12             .getProperty(VCEClientConstants.
                   VCE_PROPS_KEY_OGSADAI_RESOURCE_ID);
13         String serviceURL = properties
14             .getProperty(VCEClientConstants.VCE_PROPS_KEY_SERVICE_URL);
15
16         state = VCEClientState.STARTED;
```

```
17
18          CXFServer server = new CXFServer();
19          server.setDefaultBaseServicesURL(new URL(serviceURL));
20
21          DataRequestExecutionResource drer = server.
                getDataRequestExecutionResource(new ResourceID(
                VCEClientConstants.VCE_PROPS_VALUE_OGSADAI_DRER_ID));
22
23          Workflow workflow = WorkflowBuilder.getWorkflow(properties);
24          RequestResource requestResource = drer.execute(workflow,
                RequestExecutionType.SYNCHRONOUS);
25
26          if(workflow instanceof PipelineWorkflow){
27              Iterator<Activity> act_it = ((PipelineWorkflow) workflow).
                    getActivities().iterator();
28              while(act_it.hasNext())
29                  printActivityStatus(act_it.next());
30          }
31
32          status = requestResource.getRequestStatus();
33
34          if (status.getExecutionStatus().equals(RequestExecutionStatus.
                COMPLETED)) {
35                state = VCEClientState.FINISHED;
36          } else {
37                state = VCEClientState.ERROR;
38          }
39      }
40
41      public void getResults(File file) throws VCEException {
42        FileWriter fstream = new FileWriter(file);
43        BufferedWriter out = new BufferedWriter(fstream);
44
45        Iterator i = status.getActivityResults();
46          while (i.hasNext()) {
47            out.write(processResult(i.next().toString()));
48        }
49
50        out.close();
51      }
52  }
```

The *executeQuery* uses the keys provided by *VCEClientConstants* to extract properties from the property file. The *CXFServer* constructs the DRER instance, which represents this resource on the client-side, and builds the workflow document for it. Next, the DRER instance is executed, returning the *RequestResource* data type containing *RequestStatus*, the results of which can be diverted to a file, as described in the *getResults* method.

The Listing 6.16 shows the construction of a OGSA-DAI workflow by *WorkflowBuilder* through the *createCSVWorkflow* method. Here, a SQL data source is queried through the *SQL-Query* activity, which transforms the results to the CSV format through the *TupleToCSV* and delivers them to the *RequestStatus* through the *DeliverToRequestStatus* activity.

Listing 6.16: WorkflowBuilder.java

```
protected static Workflow createCSVWorkflow(Properties properties) throws
    VCEException {
    String queryString = properties.getProperty(VCEClientConstants.
        VCE_PROPS_KEY_SQL_QUERY);
    String resourceID = properties.getProperty(VCEClientConstants.
        VCE_PROPS_KEY_OGSADAI_RESOURCE_ID);

    SQLQuery query = new SQLQuery();
    query.setResourceID(new ResourceID(resourceID));
    query.addExpression(queryString);

    TupleToCSV tupleToCSV = new TupleToCSV();
    tupleToCSV.connectDataInput(query.getDataOutput());

    DeliverToRequestStatus deliverToRequestStatus =
          new DeliverToRequestStatus();
    deliverToRequestStatus.connectInput(tupleToCSV.getResultOutput());

    PipelineWorkflow pipeline = new PipelineWorkflow();
    pipeline.add(query);
    pipeline.add(tupleToCSV);
    pipeline.add(deliverToRequestStatus);

    return pipeline;
}
```

As discussed in Section 5.5, the *CXFServer* is the client-side representation of the server. In is described in Listing 6.17. The class implements the *Server* interface of OGSA-DAI and the *AbstractServer* class. The latter class contains functions for resolving mappings of the URL of the server to other service resources. *PortUtils* class provides a generic service stub instance, contained in the *JaxWsProxyFactoryBean* of the Apache CXF libraries.

Listing 6.17: CXFServer.java

```
public class CXFServer extends AbstractServer implements Server {
    private CXFPortUtils portUtils;

    public CXFServer() {
        mBaseURL = new URL("http://localhost:8080/dai/services/");
        portUtils = new PortUtils(this);
    }

    ...
```

```
10
11      public Resource getResource(URL url, ResourceID resourceID,
12                                  ResourceType resourceType)
13          throws ServerException, ClientToolkitException
14      {
15        if (ResourceType.DATA_REQUEST_EXECUTION_RESOURCE.equals(resourceType)
              ) {
16            throw new IllegalArgumentException(...);
17          } else if (ResourceType.DATA_RESOURCE.equals(resourceType)) {
18            return getDataResource(url, resourceID);
19          } else if (ResourceType.DATA_SOURCE.equals(resourceType)) {
20            return getDataSourceResource(url, resourceID);
21          } else if (ResourceType.DATA_SINK.equals(resourceType)) {
22            return getDataSinkResource(url, resourceID);
23          } else if (ResourceType.SESSION.equals(resourceType)) {
24            return getSessionResource(url, resourceID);
25          } else if (ResourceType.REQUEST.equals(resourceType)) {
26            return getRequestResource(url, resourceID);
27          }
28          return null;
29      }
30
31      public DataSourceResource getDataSourceResource(URL url,
32                                                      ResourceID resourceID)
33              throws ServerException, ClientToolkitException {
34          final CXFResource resource =
35                  new CXFResource(url, resourceID, ResourceType.DATA_SOURCE,
                        this);
36          return new CXFDataSourceResource(resource,this);
37      }
38
39      private ResolveResponse resolvePort(Object port, Resolve r)
40        throws uk.org.ogsadai.service.cxf.resolver.ServerFault,
              ResourceUnknownFault{
41        ResolveResponse rr = null;
42
43          if(port instanceof WSDataResourceInformationServicePortType){
44            rr = ((WSDataResourceInformationServicePortType) port).resolve(r)
                  ;
45          } else if(port instanceof WSDataRequestExecutionServicePortType){
46            rr = ((WSDataRequestExecutionServicePortType) port).resolve(r);
47          } else if(port instanceof WSDataSourceServicePortType){
48            rr = ((WSDataSourceServicePortType) port).resolve(r);
49          } else if(port instanceof WSDataSinkServicePortType){
50            rr = ((WSDataSinkServicePortType) port).resolve(r);
51          } else if(port instanceof WSRequestManagementServicePortType){
52            rr = ((WSRequestManagementServicePortType) port).resolve(r);
53          } else if(port instanceof WSSessionManagementServicePortType){
54            rr = ((WSSessionManagementServicePortType) port).resolve(r);
55          }
56
```

```
57          return rr;
58      }
59
60      private DataRequestExecutionResource
61      getDataRequestExecutionResource(URL url, ResourceID resourceID)
62              throws ServerException, ClientToolkitException {
63        final CXFResource resource =
64                  new CXFResource(url, resourceID,
65                          ResourceType.DATA_REQUEST_EXECUTION_RESOURCE, this);
66
67          return new CXFDataRequestExecutionResource(resource, this);
68      }
69  }
```

This class accesses the CXF-compliant service resources. In particular, it creates a server representations of OGSA-DAI resources, with *CXFResource* being a general representation of an OGSA-DAI Resource and *CXFDataRequestExecutionResource* being that of the DRER. *CXFResource* handles message processing through the *jax.xml.soap* classes.

The focus will be on *CXFDataRequestExecutionResource* described in Listing 6.18, which executes a web service call through the client proxy. Just as *CXFResource* adds the resource id to the message header, so does *CXFDataRequestExecutionResource* with its id to the constructor.

Listing 6.18: CXFDataRequestExecutionResource.java

```
1   public class CXFDataRequestExecutionResource
2           extends BaseDataRequestExecutionResource
3           implements DataRequestExecutionResource {
4
5       private WSDataRequestExecutionServicePortType port = null;
6       private CXFResource cxfResource;
7
8       public CXFDataRequestExecutionResource(final CXFResource resource,
            final Server server) throws Exception {
9           super(resource, server);
10          try {
11            PortUtils portUtils = ((CXFServer)server).getPortUtils();
12            cxfResource = (CXFResource) getResource();
13            port = (WSDataRequestExecutionServicePortType) portUtils.getPort(
14                ResourceType.DATA_REQUEST_EXECUTION_RESOURCE);
15
16            // Client-side headers
17            List<Header> headers = new ArrayList<Header>();
18            Header resourceIDHeader;
19            resourceIDHeader = new Header(new QName("http://ogsadai.org.uk",
                  "ResourceID"),
20            cxfResource.getResourceID().toString(), new JAXBDataBinding(
                  String.class));
21            headers.add(resourceIDHeader);
```

```
22
23          ((BindingProvider)port).getRequestContext().put(Header.
                HEADER_LIST, headers);
24      }
25      catch (Exception e) { ... }
26  }
27
28  public ExecutionResult executeRequest(
29      final ResourceID requestResourceID, final Workflow requestWorkflow,
30      final RequestExecutionType executionType,
31      final SessionOperation sessionOperation)
32      throws Exception {
33      final RequestAndStatusHandler rsh =
34          requestWorkflow.getRequestAndStatusHandler();
35
36      Execute execute = new Execute();
37
38      if (requestResourceID != null)
39          execute.setRequestID(requestResourceID.toString());
40
41      execute.setIsSynchronous(
42        executionType == RequestExecutionType.SYNCHRONOUS);
43
44      SessionRequestType session = new SessionRequestType();
45      // Session initialization...
46      ...
47      execute.setSession(session);
48
49      RequestType request = new RequestType();
50      execute.setRequest(request);
51
52      final CXFRequestBuilder rb = new CXFRequestBuilder();
53      rsh.buildRequest(rb);
54      request.setWorkflow(rb.getRootWorkflowType());
55
56      try {
57          final ExecuteResponse response = port.execute(execute);
58
59          RequestStatusBuilder requestStatusBuilder =
60                  new CXFRequestStatusBuilder(response.getRequestStatus()
                        );
61
62          final RequestStatus requestStatus =
63                  requestStatusBuilder.getRequestStatus();
64
65          if (response.getSessionID() != null) {
66              return new SimpleExecutionResult(
67                      new ResourceID(response.getRequestID()),
68                      new ResourceID(response.getSessionID()),
69                      requestStatus);
70          } else {
```

```
71                  return new SimpleExecutionResult(
72                          new ResourceID(response.getRequestID()), null,
73                          requestStatus);
74              }
75          } catch (final Exception ex){ ... }
76      }
77  }
```

The *executeRequest* method uses the *Execute* web method parameter to construct a request for the web method call, and sets *RequestType* data type. The *RequestType* then gets the workflow data type object constructed by *CXFRequestBuilder*. This concludes the parameter construction for the web method call. The *port* instance initialized in the constructor can execute the request and get *ExecuteResponse* return after the call is finished.

Further classes such as *CXFObjectSerializer*, *CXFBase64Mapper*, *CXFExceptionUtil* and *PortUtils* provide respective functional utilities to the client-side API. They, however, will not be covered here.

6.10 Service deployment tool

The deployment tool provides a graphical and a command-line interfaces to support the deployment and configuration of data services on a host. The functionality is mainly provided by Apache Ant tasks that create the web application structure, as well as configure and deploy the service into the Apache Tomcat container, together with its dependency libraries. This section will briefly cover the graphical tool for web service deployment. The two panels of Figure 6.1 show the mechanisms for deploying the service. The left panel offers the specification of the data resources. Each resource requires some configuration parameters to connect to the data source. For relational data sources, these parameters correspond to the JDBC driver parameters, whereas the DQP requires a mapping configuration for the underlying data sources.

The right panel provides configuration support of the Apache Tomcat and OGSA-DAI paths. By clicking on the 'Deploy' button, a set of tasks will start executing the service deployment to the configured Tomcat.

The structure of a deployed service is depicted in Figure B. The folder shown is deployed to the Tomcat applications container and exposed through the Internet.

The deployed data resources are located in *WEB-INF/etc/dai/resources*. In this case, there are two deployed views and the DQP resource, denoted 'ds'. Further directories of *WEB-INF/etc/dai/* include the *dqp* and *views* that contain resource configurations, while *WEB-INF/etc/classes* holds configuration of OGSA-DAI and logging features. The *WEB-INF* folder also contains the two web service configuration files *beans.xml* and *web.xml*. The JSP pages at the root

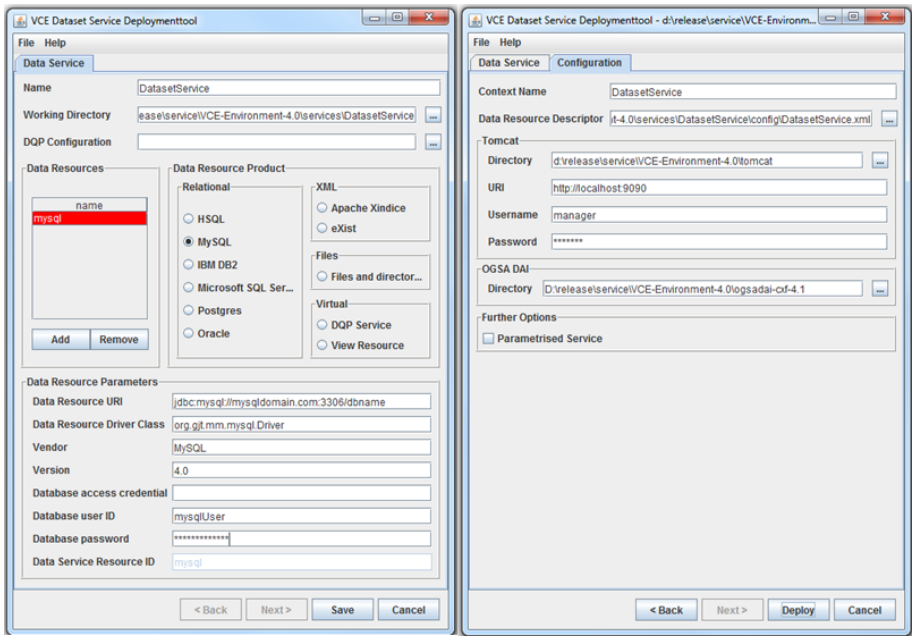


Figure 6.1: The deployment tool of VCE comprises service configuration and environment panels. The service configuration panel allows configuration of the data resources. The environment panel configures the runtime environment.

of the directory provide the OGSA-DAI web-based information pages about the service. Finally, the *schema* folder holds the WSDL descriptions.

The above description concludes the implementation of the components of the data services. Next section offers a brief description of several issues regarding the implementation and their solutions.

6.11 Issues and solutions

Several issues arose during the implementation. Table 6.1 describes these issues and solutions adopted in the implementation of the CXF web service layer for OGSA-DAI.

Table 6.1: Issues and solutions

Issue	The *ServerFactory* class checks the property file *config.properties* located in package *uk.org.ogsadai.client.toolkit.presentation* and loads the generated interface, according to the prefix given by the property *ws.wsrf.string*.
Solution	Since the generated interface has the prefix *WS*, e.g. *WSDataRequestExecutionServicePortType*, the property value needs to be adjusted to *WS*.

Issue	The *AbstractServer* class method *getDefaultRequestManagementServiceURL()* returned the wrong URL for the service resources.
Solution	Since this method should return the URL for the Request Management Service, the return statement *return getDefaultServiceURL(ResourceType.SESSION);* is changed to *return getDefaultServiceURL(ResourceType.REQUEST);*
Issue	OGSA-DAI uses the client-toolkit classes to establish communication between services. When using DQP the services propagate the URL without *?wsdl* at the end of that URL. This is caused by a bug.
Solution	A simple workaround in the *getServer()* method verifies if *?wsdl* is at the end of the URL, and adds it if missing.
Issue	The method *createMessageElementForProperty()* in class *CXFWSResourcePropertiesProvider* was not properly adding property elements to the SOAP message.
Solution	The method was rewritten and modified to utilize JAX-WS classes to add content to the SOAP message.
Issue	In several locations throughout the software the URLs require the transformation to their canonical form.
Solution	Use the *toExternalForm()* method of the URL class to get the canonical URL.
Issue	Casting an array in *listResources String[] resourceIdStrings = (String[]) lrr.getResourceID().toArray();* causes a *ClassCastException* due to the fact that the *toArray* mathod returns a *List* not a *List<String>*
Solution	changing the procedure to copy the array with *to Arrays.copyOf*: *String[] resourceIdStrings = Arrays.copyOf(lrr.getResourceID().toArray(), lrr.getResourceID().toArray().length, String[].class);* corrects this issue.
Issue	Due to strict type-checking by the JAX-WS specification of declared types in the WSDL, it is no longer possible to assign the the PipeID, in form of String, as the connector. The *set*Stream* method of **StreamType* requires an object of a **StreamType*, while *setPipe* method of the same class is used to set the *PipeID*.
Solution	A Hashmap is used to map *PipeIDs* to the corresponding **StreamType* objects in the *CXFRequestBuilder* class.
Issue	While both OGSA-DAI and the Spring Framework utilize the XML parser Apache Xerces, OGSA-DAI uses an older version. Both libraries are deployed by default, which results in a class conflict.
Solution	This issue is resolved by taking the older Xerces version out of compilation and deployment.
Issue	Due to the prototype implementation of basic HTTP authentication in the remote cloud, the services had to be adjusted to authenticate themselves at the point of client proxy creation (because the service WSDLs could not be retrieved). The service implementation had the same issue, since data services were not able to communicate with the federation service.

Solution	A temporary solution was adopted using the *java.net.Authenticator* class for transport layer security and a *CXFAuthenticator* was implemented to enable this kind of authentication. A permanent solution will be implemented using the Apache CXF *HTTPConduit* class for handling authenticated HTTP communications.
Issue	Another issue arose during the implementation of the CXFAuthenticator. The Base64 encoding of the security token in Java 1.6 would take not more that 76 characters and results in an illegal character exception. The token provided to access the remote endpoint URL has more than that.
Solution	As of Java 1.7 this issue is resolved. Servers and clients where updated and tested accordingly.
Issue	The remote cloud uses web server proxies to relay deployed VMs into a sub-domain. Since the data service spring configuration file *beans.xml* declares to URL mappings as a relative path, these mappings will not work behind a proxy.
Solution	The deployment tool has a field for entering a custom server URL.

6.12 Complexity

The CXF specific layer has three source folders:

- 'server' having 18 packages, 27 classes with approximately 3500 lines of code
- 'client' having 8 packages, 13 classes with approximately 2800 lines of code
- 'proxy' having 35 packages, 155 class with approximately 7000 lines of code

In total, there are over 60 packages and 13000 lines of code, not counting blank lines or comments. Further complexity metric analysis concludes that the implementation has over 800 methods and 220 types. Most of them, however, were generated for the proxy implementation.

6.13 Summary

This section provided the description of the implementation for the data service, including definitions, implementation classes and configuration. The final chapter will conclude this work with an experimental evaluation of the services based on the scenarios specified in Section 4.8.

Chapter 7

Experimental evaluation

The best performance improvement is the transition
from the non-working state to the working state.
–John Ousterhout

This final chapter runs the scenarios described in Section 4.8. It offers a detailed description of the setup of the data services, showcasing performance and scenario results. Understanding the performance of a distributed system of this type is not an easy task. Because of remotely located data services, the performance of the system will depend on the current bandwidth, which may change in the course of a day. The presented results are meant to give the reader an idea of what one may expect when running requests through the data infrastructure, and compare the old and the new implementation.

7.1 Testing and evaluation strategy

The main objective for performance testing of the data services is to provide a measurement of the query time utilizing both the federated and direct services to retrieve the data. Data services will have a data transfer time, while the federated approach will have of an additional overhead in terms of the DQP evaluation time.

As described in Section 4.8, the first scenario will provide a comparison between the Apache Axis and CXF presentation layer implementations. This serves two purposes. It shows that the system functions as designed, and is more efficient than the existing implementation. The improvement in query time, further referred to as speedup, and the DQP overhead will be calculated and compared to that of the existing implementation. The second test scenario will compare the performance of the federation service in different cloud environments. The three

types of cloud environments tested include:

- Local private cloud, where a federation service runs on the same network as the data services located in Vienna
  - the VM in the local cloud is assigned 2 VCPU cores and 1GB memory;
- Remote private cloud, where the federation service runs on a remote network in Poland
  - the VM in the remote cloud is assigned 1 VCPU cores and 0.5GB memory;
- Commercial cloud, where the federation service runs on Amazon EC2 instances in Ireland
  - the specification of VMs in the commercial Amazon EC2 cloud can be viewed in [67].

An important question is whether the query execution time is largely determined by data transfer time, or whether more expensive and better equipped instances for a distributed query service can improve the performance.

Each test will be carried out 30 times, recording average and median times. The first two executions will be excluded, as measurements may occur where the OGSA-DAI context is not yet fully initialized. The tests will be executed on the same day in a time span of under two hours. This is done to minimize the influence of bandwidth variations on performance.

7.2 Setup

The current release of the data services uses OGSA-DAI core version 4.1, Apache CXF 2.4.1 and Tomcat 7.0.25 with Java JDK 7u11 on CentOS 6.3. The JRE will have 256MB of memory. Extensions included in OGSA-DAI are DQP, RDF and Views. The tests have been implemented in JUnit 4.

The data services are deployed on two physical machines, each having an Intel Xeon X3360 processor with 4GB of memory, 500GB HDD and 10/100/1000MB network switch, while the federation services will be deployed on virtual machines as part of virtual data appliances. Data services will expose a MySQL 5.5 community edition DBMS deployed on the same physical machine.

As mentioned in Section 4.8, the tests are carried out on two sets of clinical data. The problem, however, is that both datasets are rather small. Combined, they have less than 500 rows. This is certainly not enough for a thorough testing. To circumvent the shortage of real

data, larger sets of random data have been generated to mimic the real data. The client have been set to request a synchronous delivery of data in the *WebRowSet* XML format, which will further inflate the returned data. Three classes of data sizes listed below have been tested:

The data transfer sizes are divided into 3 classes:

- small - 1000 rows, 497.211 bytes (485 KB)
- medium - 10000 rows, 2.334.666 bytes (2.22 MB)
- large - 100000 rows, 18.980.031 bytes (18.1 MB)

The data schema comprises three tables, two of which come from a patient database for treating aneurisms, and the clinical reference information model (CRIM). The other (PVP) extends patient data from a general treatment department of a clinic. The generated tables will have the following schema:

- crim.root
  - id - primary key, integer
  - patient - foreign key, patiend id, integer
  - weight - float
  - height - float
  - brainTumor - varchar(255)
- crim.aneudetailevent
  - id - primary key, integer
  - patient - foreign key, patiend id, integer
  - type - varchar(255)
  - location - varchar(255)
  - aspect - varchar(255)
  - rupture - varchar(255)
  - side - varchar(255)
- pvp.patients
  - id - primary key, integer
  - procedure - varchar(255)
  - status - varchar(255)

- review - varchar(255)
- sex - varchar(255)
- smoker - varchar(255)
- deceased - varchar(255)

The CRIM and PVP tables will be exposed by separate data services.

7.3 Implementation scenario results

This scenario provides a comparison between two implementations of the presentation layer. The speedup, that represents the improvement of the query time, is calculated by dividing the average of the two implementations. The execution time is given in milliseconds.

Table 7.1 shows results of query execution directly to the data services exposing the data sources. The speedup factor is computed as the ratio of the average run times of Apache CXF to Apache Axis. The results of the test clearly show that the speedup factor (7.71 given a large data set) increases with the amount of data.

Table 7.1: Data service comparison of presentation layer implementations

Data size	Axis		CXF		Speedup factor
	Average	Median	Average	Median	
Large	13879	13922	1799	1794	7.71
Medium	520	455	199	186	2.61
Small	104	94	132	109	0.79

Here, the comparison of the data service performance with respect to the old (Apache Axis) and the new (Apache CXF) implementation is presented. The time is provided in milliseconds and the speedup represents the improvement over the query execution time.

While the results of the first test are promising already, the second table 7.2 benchmarks the federation services in the two implementations. As before, the table shows the increase in the speedup factor, as the amount of data increases. Note that while both implementations start processing data stream as it arrives, the Apache CXF implementation has the larger overhead than the Apache Axis presentation layer implementation, because the computational tasks of the process cannot keep up with the speed of data transfer. This suggests that data transfer rate is not a bottleneck in the CXF implementation. The Axis implementation does not have this advantage, and this shows in higher overhead times. Of course, the overhead comparisons are conducted within the same implementation.

Table 7.2: Federated service comparison of presentation layer implementations

Data size	Axis		CXF		Speedup factor	Overhead Modifier	
	Average	Median	Average	Median		Axis	CXF
Large	15424	15093	3591	3418	4.3	0.90	0.50
Medium	839	830	558	546	1.5	0.62	0.36
Small	399	385	404	423	1.0	0.26	0.32

This scenario compares performance of the old (Apache Axis) and new (Apache CXF) implementations through the federation service. In addition to the speedup, the overhead modifier provides a factor of additional costs it takes for the federation service to collect data.

7.4 Federation scenario results

The next scenario provides an insight into the performance of the federation service on different clouds. The local federation service in Vienna is used for comparison of the remote federation services. While the previous tables were sorted by performance, table 7.3 is sorted by instance hourly fee of the Amazon EC2 instances. This provides the reader with a measurement of performance in relation to the total fee.

While the additional overhead between the local and remote federation services is minimal, the overhead of the Amazon EC2 instance is fairly significant. This was expected, since typically a dedicated private cloud has less traffic than a commercial one. Nevertheless, the question whether more expensive and better equipped instances for a distributed query service can improve the performance can now be answered. While the process is mainly driven by the transfer-rate, there is only so much a better equipped instance can do. For example, the relatively cheap high-computing instance c1.xlarge has showed the fastest execution time. Standard instances, on the other hand, do not provide a good host because they have limited computational capabilities.

One surprising result in this benchmark came from the hi1.4xlarge instance. It is an expensive cluster-type instance designed for I/O-intensive applications, providing a high bandwidth and low latency connection. Unfortunately, it was not at all a top performer. This may be due to the size of the dataset, as this instance is provided for 'Big Data'-type applications. The other reason might lie in the working memory assigned to the JRE, which is held to 256MB throughout all instances. While this setting might be too small, it is consistently used to provide comparable results in all tests.

Nevertheless, the answer to the question posed in Section 7.1 is a tentative 'no'. A better approach to improving the performance is to seek for a balance in CPU capacity, memory and

bandwidth for the data services, rather than host them on better equipped but more expensive hosts. This could be achieved through implementation of an Auto-Tuning feature, that sets the runtime environment parameters according to the host properties.

Table 7.3: Cloud-related data federation scenario

Instance	Large		Medium		Small		$ Price/hour
	Average	Median	Average	Median	Average	Median	
Local	3591	3418	558	546	404	423	
Remote	4839	4574	1160	1164	952	945	
EC2 m1.small	17410	17502	2257	2129	1316	1264	0.085
EC2 m1.medium	14797	14752	1883	1865	1204	1148	0.170
EC2 c1.medium	12894	12701	1861	1748	1130	1094	0.186
EC2 m1.large	12771	12683	1715	1665	1120	1101	0.340
EC2 m2.xlarge	13239	13425	1600	1552	1091	1054	0.506
EC2 m1.xlarge	14780	14919	1854	1808	1210	1138	0.680
EC2 c1.xlarge	11871	11747	1589	1570	1099	1076	0.744
EC2 m2.2xlarge	14709	14777	1760	1666	1102	1080	1.012
EC2 m2.4xlarge	12731	12638	2257	2129	1316	1264	2.024
EC2 hi1.4xlarge	14780	14919	1854	1808	1210	1138	3.410

This scenario tests the performance of the federation services in several cloud environments. The local (Vienna) instance serves as a reference, since it is in the same cloud as the data services. The remote instance is placed in a dedicated private cloud in Poland, while the following instances are deployed in the Amazon EC2 cloud. The list is sorted by price.

Figure 7.1 depicts the results presented in Table 7.3, sorting them by query execution time. The results confirm that high-CPU instances are a better choice in contrast to high-memory instances.

7.5 Conclusions and future plans

This thesis presented a data service infrastructure in context of the VPH-Share project. The data infrastructure is based on the composition of (i) static data services that expose clinical data sources and (ii) dynamic federation services that combine them to provide a higher level of abstraction for the data infrastructure. The data service architecture comprises several components, a prominent one being OGSA-DAI. OGSA-DAI allows wrapping the data resource and exposing it as a web service. The web service data integration is thus accomplished through OGSA-DAI's presentation layer. However, the SOAP-based OGSA-DAI release is based on

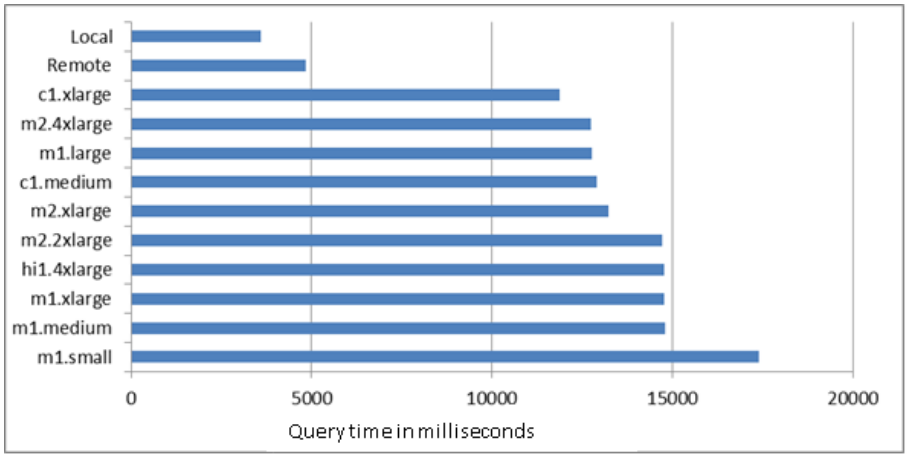


Figure 7.1: This chart shows the results of the federation evaluation scenario. Results are sorted by query execution time.

the outdated Apache Axis web service framework is not up to the requirements of supported state-of-art web service standards and performance. Due to these drawbacks of the existing Apache Axis web service framework, a new SOAP-based presentation layer for OGSA-DAI based on Apache CXF web service framework was developed.

This thesis presented the software realization process to plan, design, implement and evaluate the new presentation layer. The planning phase has identified the requirements and primary use cases for the presentation layer, while discussing testing scenarios; the designing part of this work concerned itself with the modeling of the presentation layer and presented sequence diagrams for the use cases and a class diagram for the implementation, explaining how the architecture of OGSA-DAI allows to substitute the presentation layer. The implementation layer provided the source code parts critical for creation of the new presentation layer and described issues during the implementation and their tentative solutions.

This chapter presented performance and test results in relation to the implementation and deployment of federation services in remote clouds. The tests showed an improvement over the older presentation layer, but there is still room for more. In particular, the provisional security solution, described in Section 6.11 has to be implemented properly. For additional performance, a detailed profiling of OGSA-DAI may be necessary to identify bottlenecks in the middleware layer. The goal would be to relay data processing as near to the data source as possible.

Further improvements to the application are planned to be implemented. As stated in Section 4.6, VCE requires a RESTful interface in addition to SOAP messaging. This is currently being implemented. In addition, to provide the querying over multiple semantic sources, the SPARQL Federation Extension, similar to the relational DQP, is high on the agenda. Further improvements may be considered in relation to performance tuning, as stated in Section 7.4. The Auto-Tune feature would then be able to adapt the application setup to the hosting environ-

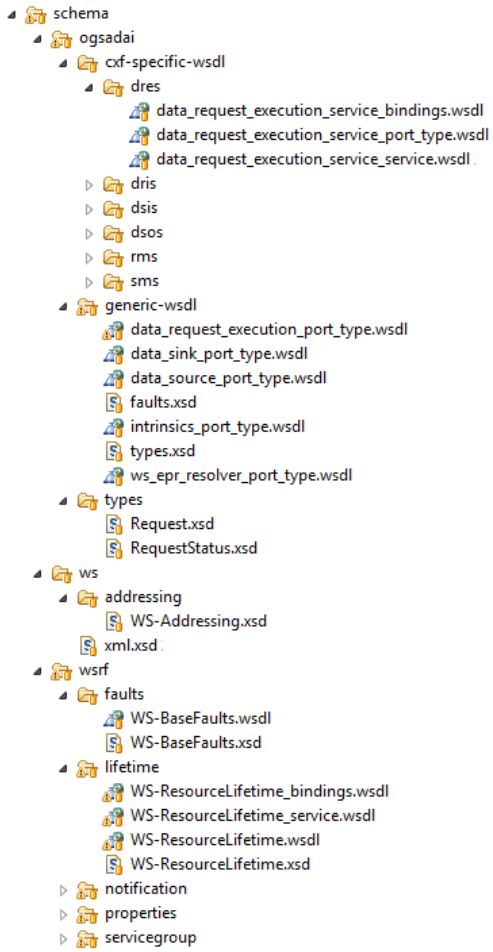
ment and exploit the given resources more efficiently.

The OGSA-DAI's latest released is based on Jersey Technology, which implements a RESTful communication pattern and omits SOAP messaging. Should the OGSA-DAI development team consider this work an appropriate contribution to their software release, establishing contact and transferring the knowledge described in this work to the OGSA-DAI development team will be considered, after fixing temporary issues described in Section 6.11.

Appendix A

WSDL definition stucture of OGSA-DAI

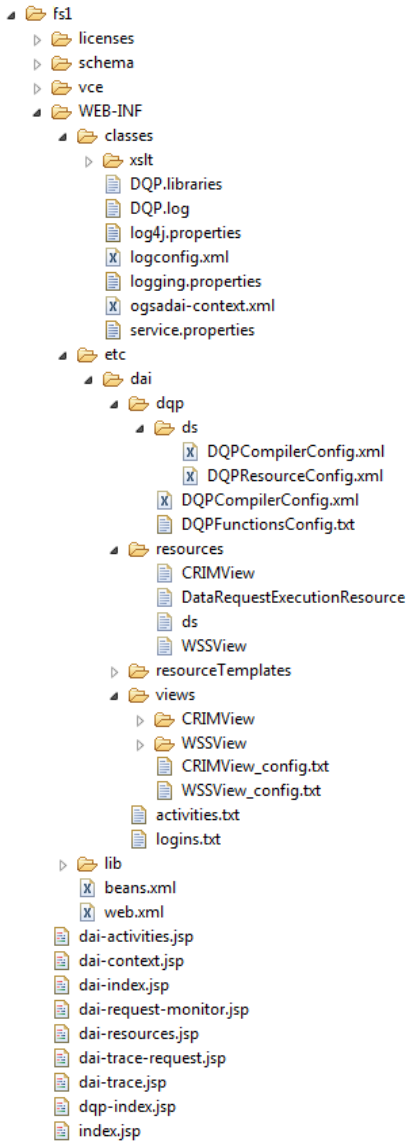
This figure shows the schema directory structure of OGSA-DAI. The three main directories divide definitions for OGSA-DAI, WS-Addressing and WSRF extensions. In them, each separate service and extension property is strictly specified.



Appendix B

Deployed data service directory structure

The following figure shows the resulting directory structure of a deployed data service.



Bibliography

[1] M. Antonioletti, M. Atkinson, R. Baxter, A. Borley, C. Hong, P. Neil, B. Collins, N. Hardman, A.C. Hume, A. Knox, M. Jackson, A. Krause, S. Laws, J. Magowan, N.W. Paton, D. Pearson, T. Sugden, P. Watson, M. Westhead, The design and implementation of grid database services in ogsa-dai: Research articles. *Concurrency and Computation : Practice and Experience* **17**(2-4), 357 (2005)

[2] M.P. Papazoglou, et al., Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal* (2007)

[3] VPH. VPH for researchers, http://www.vph-noe.eu/objectives/vph-for-researchers?showall=1 (2012). Accessed: 15/02/2012

[4] M. Köhler, S. Benkner, VCE - A Versatile Cloud Environment for Scientific Applications, in *The Seventh International Conference on Autonomic and Autonomous Systems (ICAS 2011)* (XPS, Venice-Mestre, Italy, 2011)

[5] P. Nowakowski, T. Bartynski, T. Gubala, D. Harezlak, M. Kasztelnik, M. Malawski, J. Meizner, M. Bubak, Cloud Platform for VPH Applications, in *8th International Conference on eScience 2012* (IEEE, Chicago, USA, 2012)

[6] I. Janciak, C. Kloner, P. Brezany, Workflow enactment engine for WSRF-compliant services orchestration, in *9th IEEE/ACM International Conference on Grid Computing (Grid 2008), Tsukuba, Japan, September 29 - October 1, 2008* (IEEE, 2008), pp. 1–8

[7] R.T. Fielding, Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California, Irvine (2000). AAI9980887

[8] M. Ganzinger, T. Noack, S. Diederichs, T. Longerich, P. Knaup, Service oriented data integration for a biomedical research network. *Stud Health Technol Inform* **169** (2011)

[9] M. Lenzerini, Data integration: a theoretical perspective, in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (ACM, New York, NY, USA, 2002), PODS '02, pp. 233–246

[10] A.P. Sheth, J.A. Larson, Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.* **22**(3), 183 (1990)

[11] Z.G. Ives, Efficient Query Processing for Data Integration. Ph.D. thesis, University of Washington, Irvine (2002)

[12] A.Y. Levy, Logic-based techniques in data integration, in *Logic-based artificial intelligence*, ed. by J. Minker (Kluwer Academic Publishers, Norwell, MA, USA, 2000), pp. 575–595

[13] J. Ullman, Information integration using logical views, in *Database Theory ICDT '97*, *Lecture Notes in Computer Science*, vol. 1186, ed. by F. Afrati, P. Kolaitis (Springer Berlin / Heidelberg, 1997), pp. 19–40

[14] S. Garfinkel. The Cloud Imperative,
http://www.technologyreview.com/news/425623/the-cloud-imperative/
(2011). Accessed: 10/10/2011

[15] U. Banerjee. Cloud Computing Important Events till 2010,
http://setandbma.wordpress.com/2011/03/08/cloud-computing-important-events-till-2010
(2011). Accessed: 20/09/2011

[16] N. Carr, *The Big Switch: Rewiring the World, from Edison to Google* (W.W. Norton & Company, 2009)

[17] D. Parkhill, *The Challenge of the Computer Utility*. No. p. 246 in The Challenge of the Computer Utility (Addison-Wesley Pub. Co., 1966)

[18] XSEDE. TeraGrid Project Archives,
https://www.xsede.org/tg-archives
(2011). Accessed: 02/12/2011

[19] EGEE. TeraGrid Project Archives,
http://www.eu-egee.org/
(2011). Accessed: 03/12/2011

[20] Jeff Widman. Salesforce.com Launches The Service Cloud, A Customer Service SaaS Application,
http://techcrunch.com/2009/01/14/salesforcecom-launches-the-service-cloud-a-customer-service-saas-application/
(2009). Accessed: 15/01/2013

[21] L.A. Barroso, J. Dean, U. Holzle, Web search for a planet: The Google cluster architecture. *Micro, IEEE* **23**(2), 22 (2003)

[22] Amazon. Amazon EC2 Beta,
http://aws.typepad.com/aws/2006/08/amazon_ec2_beta.html
(2006). Accessed: 15/01/2013

[23] Microsoft. IaaS, PaaS und SaaS,
http://www.microsoft.com/austria/enterprise/article.aspx?id=IaaS+PaaS+und+SaaS
(2011). Accessed: 23/09/2011

[24] OASIS. Organization for the Advancement of Structured Information Standards,
http://www.oasis-open.org
(2011). Accessed: 22/09/2011

[25] W3C. Software oriented Architecture,
http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/
(2004). Accessed: 11/10/2012

[26] P. Fremantle, S. Weerawarana, R. Khalaf, Enterprise services. *Communications of the ACM* **45**(10), 77 (2002)

[27] G. Alonso, F. Casati, H. Kuno, V. Machiraju, *Web Services: Concepts, Architectures and Applications* (Springer, Berlin, 2004)

[28] W3C. SOAP 1.2 Specification,
http://www.w3.org/TR/soap/
(2007). Accessed: 11/10/2012

[29] W3C. WSDL 2.0 Specification,
http://www.w3.org/TR/wsdl20/
(2007). Accessed: 11/10/2012

[30] OASIS. UDDI 1.2 Specification,
https://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm
(2007). Accessed: 11/10/2012

[31] W3C. Message Transmission Optimization Mechanism (MTOM),
http://www.w3.org/TR/soap12-mtom/
(2005). Accessed: 11/10/2012

[32] PyWPS. WSDL,
http://wiki.rsg.pml.ac.uk/pywps/WSDL
(2012). Accessed: 21/04/2012

[33] S. Tilkov. UDDI R.I.P,
http://www.innoq.com/blog/st/2010/03/uddi_rip.html
(2010). Accessed: 11/10/2012

[34] Athens University of Economics and Business, Department of Management Science & Technology. Web Service Technologies,
http://www.dmst.aueb.gr/dds/etech/arch/wsa_tech.htm
(2003). Accessed: 02/04/2011

[35] Wikipedia. List of web service specifications,
http://en.wikipedia.org/wiki/List_of_Web_service_specifications
(2012). Accessed: 11/10/2012

[36] SRB team (The University of North Carolina at Chapel Hill). What is the SRB,
http://www.sdsc.edu/srb/index.php/What_is_the_SRB
(2006). Accessed: 16/02/2012

[37] E. Ambrosi, A. Ghiselli, G. Taffoni, GDSE: a new data source oriented computing element for the Grid, in *Proceedings of the 24th IASTED international conference on Parallel and distributed computing and networks* (ACTA Press, Anaheim, CA, USA, 2006), PDCN'06, pp. 53–57

[38] IBM. IBM WebSphere Information Integrator,
http://www-01.ibm.com/support/docview.wss?uid=swg24026174
(2012). Accessed: 27/04/2012

[39] H. Shannon, L. Stephen, O. Scott, S. Joel, Distributed Data Management and Integration: The Mobius Project, in *Proceedings of the Global Grid Forum 11 (GGF11) Semantic Grid Applications Workshop* (2004), pp. 20–38

[40] M. Atkinson, P. Brezany, O. Corcho, L. Han, J. van Hemert, L. Hluchy, A. Hume, I. Janciak, A. Krause, D. Snelling, A. Wöhrer, Advanced Data Mining and Integration Research for Europe, in *All Hands Meeting 2009* (Oxford, 2009)

[41] S. Benkner, A. Arbona, G. Berti, A. Chiarini, R. Dunlop, G. Engelbrecht, A.F. Frangi, C.M. Friedrich, S. Hanser, P. Hasselmeyer, R.D. Hose, J. Iavindrasana, M. Köhler, L.L. Iacono, G. Lonsdale, R. Meyer, B. Moore, H. Rajasekaran, P.E. Summers, A. Wöhrer, S. Wood, @neurIST: Infrastructure for Advanced Disease Management Through Integration of Heterogeneous Data, Computing, and Complex Processing Services. *Information Technology in Biomedicine, IEEE Transactions on* **14**(6), 1365 (2010)

[42] Department of Electrical and Electronic Engineering, Imperial College London. Mobile Environmental Sensing System Across a Grid Environment (MESSAGE),
http://www.commsp.ee.ic.ac.uk/ wiser/message/
(2011). Accessed: 02/12/2011

[43] Berry, Djaoui, Grimshaw, Horn, Maciel, Siebenlist, Subramaniam, Treadwell, V. Reich, .J. 2006. The Open Grid Services Architecture, Version 1.5 (2006)

[44] I. Foster, C. Kesselman, J. Nick, S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration (2002)

[45] M. Antonioletti, A. Krause, N.W. Paton, A. Eisenberg, S. Laws, S. Malaika, J. Melton, D. Pearson, The WS-DAI family of specifications for web service data access and integration. *SIGMOD Rec.* **35**(1), 48 (2006)

[46] M.N. Alpdemir, A. Mukherjee, A. Gounaris, N.W. Paton, P. Watson, A.A. Fernandes, D.J. Fitzgerald, OGSA-DQP: A Service for Distributed Querying on the Grid, in *Advances in Database Technology - EDBT 2004*, *Lecture Notes in Computer Science*, vol. 2992, ed. by E. Bertino, S. Christodoulakis, D. Plexousakis, V. Christophides, M. Koubarakis, K. Böhm, E. Ferrari (Springer Berlin / Heidelberg, 2004), pp. 3923–3923

[47] S. Lynden, A. Mukherjee, A.C. Hume, A.A.A. Fernandes, N.W. Paton, R. Sakellariou, P. Watson, The design and implementation of OGSA-DQP: A service-based distributed query processor. *Future Generation Computer Systems* **25**(3), 224 (2009)

[48] I. Kojima, Design and Implementation of OGSA-DAI-RDF, in *GGF16 Semantic Grid Workshop*. (Athens, Greece, 2006)

[49] C. Buil-Aranda, Federated Query Processing for the Semantic Web. Ph.D. thesis, Departament of Artificial Intelligence, Technical University of Madrid (UPM) (2012)

[50] D. Benslimane, S. Dustdar, A. Sheth, Services Mashups: The New Generation of Web Applications. *IEEE Internet Computing* **12**(5), 13 (2008)

[51] java.net. JAX-WS Reference Implementation,
http://jax-ws.java.net/
(2012). Accessed: 15/10/2012

[52] Ijeoma N. Mba, Matthew O. Adigun, Comparative Study of Web Services Platforms in a GUISET Environment (2011). Working paper of the University of Zululand

[53] W3C. Resource Description Framework,
http://www.w3.org/RDF/
(2004). Accessed: 13/11/2012

[54] D. Sosnoski. Java web services: CXF performance comparison,
http://www.ibm.com/developerworks/opensource/library/j-jws14/index.html?ca=drs-
(2010). Accessed: 22/09/2011

[55] J. Sevellec. Bench de framework web service java: Axis2 vs Cxf vs Spring ws,
http://www.unchticafe.fr/2010/02/bench-de-framework-web-service-java.html
(2010). Accessed: 22/09/2011

[56] The OGSA-DAI team. OGSA-DAI binary transfer benchmark,
http://sourceforge.net/apps/trac/ogsa-dai/wiki/BinaryTransfer
(2010). Accessed: 15/07/2011

[57] Spring. The Spring Framework - Reference Documentation,
http://static.springsource.org/spring/docs/2.5.4/reference/index.html
(2012). Accessed: 31/07/2012

[58] The OGSA-DAI team. OGSA-DAI Java Coding Guidelines,
http://sourceforge.net/apps/trac/ogsa-dai/wiki/RepositoryCodingGuidelines
(2011). Accessed: 12/04/2011

[59] I. Foster, Globus toolkit version 4: Software for service-oriented systems, in *IFIP International Conference on Network and Parallel Computing* (Springer-Verlag, 2005), no. 3779 in LNCS, pp. 2–13

[60] The OGSA-DAI team. OGSA-DAI Resource specification,
http://ogsa-dai.sourceforge.net/documentation/ogsadai4.1/ogsadai4.1-axis/ResourcesSpecification.html
(2005). Accessed: 12/02/2012

[61] The OGSA-DAI team. OGSA-DAI Architecture,
http://www.globus.org/toolkit/docs/development/3.9.5/techpreview/ogsadai/doc/background/architecture.html
(2005). Accessed: 12/02/2012

[62] The OGSA-DAI team. OGSA-DAI Services,
http://ogsa-dai.sourceforge.net/documentation/ogsadai4.1/ogsadai4.1-axis/Services.html
(2009). Accessed: 08/05/2012

[63] K. Pradeeban. OGSA-DAI Presenting in CXF,
http://kkpradeeban.blogspot.co.at/2010/06/ogsa-dai-presenting-in-cxf.html
(2010). Accessed: 01/03/2012

[64] The OGSA-DAI team. CXF Issues,
http://sourceforge.net/apps/trac/ogsa-dai/ticket/321
(2011). Accessed: 01/03/2012

[65] The OGSA-DAI team. OGSA-DAI and Subversion,
http://sourceforge.net/apps/trac/ogsa-dai/wiki/OGSADAISubversion
(2012). Accessed: 01/03/2012

[66] R. Butek. Which style of WSDL should I use?,
http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/
(2005). Accessed: 22/04/2012

[67] Amazon Web Services. Amazon EC2 Instance Types,
http://aws.amazon.com/en/ec2/instance-types/
(2012). Accessed: 24/10/2012