



universität
wien

MASTERARBEIT

Titel der Masterarbeit:

**MANAGEMENT OF AND INTERACTION WITH OLAP
CLOUD SERVICE**

eingereicht von:

Sicen Ye

zur Erlangung des akademischen Grades

Diplom-Ingenieur(Dipl.-Ing.)

Wien, October 2011

Matrikelnummer: 0309037

Studienrichtung: Scientific Computing A066 940

Begutachter: Ao. Univ.-Prof. Dipl.-Ing. Dr. Peter Brezany

Ich versichere:

- dass ich die Diplomarbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe.
- dass ich diese Diplomarbeit bisher weder im In- noch im Ausland (einer Beurteilung bzw. einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt habe.
- dass diese Arbeit mit der vom Begutachter beurteilten Arbeit bereinstimmt.

Wien, October 2011

Sicen Ye

Abstract

Cloud Computing is a relatively newly emerged high performance parallel computing paradigm. A lot of algorithms from the past could now find new opportunities and benefit from it. After several month of study on theory and implementation of On-Line Analytical Processing (OLAP), especially, the OLAP engine from the Grid-Miner project (<http://www.gridminer.org>), we decided to design and implement an OLAP system for Cloud Computing environment. For this cloud-enabled OLAP system we have also provided means for management and interaction with it, which are implemented by a multi-tier client subsystem including some business logic and Graphical User Interfaces (GUI) in an easy to use and understandable way. In this thesis the original design and implementation of the multi-tier client subsystem is described and discussed.

Management and interaction with OLAP cloud means on one hand loading data from data source, transforming and transferring it to the OLAP cloud to construct data cube, On the other hand, submitting OLAP analysis queries and handling the results.

Practically, the implemented client subsystem was developed mainly using Google Web Toolkit (GWT) as a web-based multi-tier application. It is able to load data either from a single Relational Database Management System (RMDBS) via Java Database Connectivity (JDBC), or from Open Grid Services Architecture - Data Access and Integration (OGSA-DAI) server, which integrates data from heterogeneous data sources. Operations such as loading data to OLAP cloud and OLAP query are achieved by interacting with the Representational State Transfer (REST) APIs provided by the OLAP cloud. Data is represented in WebRowSet format, operations are described in OLAP Modeling Markup Language (OMML) version 2.0, which is proposed, described and implemented in this thesis.

Zusammenfassung

Cloud Computing ist ein relativ neu entstandenes High Performance Parallel Computing Paradigma. Viele Algorithmen aus der Vergangenheit können nun von Cloud Computing profitieren und neue Möglichkeiten finden. Nach einigen Monaten Untersuchungen zur Theorie und Umsetzung von On-Line Analytical Processing (OLAP), vor allem, die OLAP-Engine aus dem GridMiner Projekt (<http://www.gridminer.org>), haben wir beschlossen, ein OLAP System für Cloud Computing Umgebung zu entwerfen und Implementieren. Für das Cloud-enabled OLAP System brauchen wir auch Mittel für das Management und Interaktion mit ihm, die durch eine Multi-Tier Client Subsystem einschließlich einiger Business-Logik und Graphical User Interfaces (GUI) umgesetzt werden sollten. In dieser Arbeit die ursprüngliche Gestaltung und Umsetzung der Multi-Tier Client Subsystem wird beschrieben und diskutiert.

Management und Interaktion mit dem OLAP Cloud bedeutet auf der einen Seite das Laden von Daten aus der Datenquelle, die Umwandlung und Übertragung von Daten auf das OLAP Cloud um Data Cube zu konstruieren, Auf der anderen Seite, Versand von OLAP Analyse Abfragen und Empfang des Ergebnis.

Unser Client-Subsystem wurde hauptsächlich mit dem Google Web Toolkit (GWT) als web-basierte Multi-Tier Anwendung entwickelt. Es könnte Daten von verschiedenen Datenquelle einlesen, z.B. von Relational Database Management Systeme (RMDBS) oder von integrierten Abfrage-Ergebnis von mehreren RMDBS durch Open Grid Services Architecture - Data Access and Integration (OGSA-DAI) Server, der Distributed Query Processing bietet (DQP). Operationen wie das Laden von Daten auf OLAP Cloud und OLAP Abfrage sind durch die Interaktion mit dem Representational State Transfer (REST) APIs des OLAP Cloud erreicht. Daten in unserem System werden in WebRowSet Format dargestellt, und die Operationen werden mit OLAP Modelling Markup Language (OMML) version 2.0 beschreibt, die in dieser Masterarbeit vorgeschlagen, beschrieben und implementiert wird.

Contents

Abstract	v
Zusammenfassung	vii
List of Figures	xi
List of Tables	xiii
1 Introduction	1
1.1 Motivation	1
1.2 Goals	2
1.3 Approach	5
1.4 Thesis Organization	6
2 Basics of On-Line Analytical Processing	8
2.1 Basic Principles	8
2.2 Classification of OLAP	11
2.3 MOLAP Operations	12
2.3.1 Aggregation	12
2.3.2 Roll-up and Drill-down	12
2.3.3 Slice and Dice	14
2.3.4 Pivot	15
3 Design and Implementation	17
3.1 Multi-tier Architecture	17
3.1.1 Design of the Service Tier	18
3.1.2 Design of the GWT Client Tier	18
3.1.3 Design of the GWT Server Tier	18
3.2 Data Flow in the System	19
3.2.1 Data Flow of the OLAP Query Client	19
3.2.2 Data Flow of the OLAP Administrator	20
3.3 Why Multi-tier Architecture	21
3.4 Implementation	23
3.4.1 Implementation of OLAP Access Servlet	23
3.4.2 Implementation of Database Access Servlet	28
3.4.3 Implementation of OGSA-DAI Access Servlet	31
3.4.4 Implementation of the WebRowSet XML Generator	40

3.4.5	Convert the WebRowSet XML Document to Ext GWT Grid	43
3.4.6	Modules in GWT Development	45
3.4.7	Implementation of PRC Call	47
3.5	UML Diagrams Description	50
3.5.1	UML Diagram of the OLAP Query Client Project	51
3.5.2	UML Diagram of the OLAP Administrator Project	54
4	Installation and Deployment	58
4.1	Preparation	58
4.2	Installation	59
4.2.1	Installation for Both OLAP Query Client and OLAP Admin- istrator	59
4.2.2	Installation for OLAP Query Client	62
4.2.3	Installation for OLAP Administrator	66
4.3	Deployment	70
4.3.1	Deployment for OLAP Query Client	70
4.3.2	Deployment for OLAP Administrator	71
5	Graphical User Interface	76
5.1	Description of Testing Data Set	76
5.1.1	Prerequisites	77
5.2	Introduction of OLAP Administrator GUI	77
5.3	Introduction of OLAP Query Client GUI	80
6	OLAP Modelling Markup Language	82
6.1	The Components of OMML	82
6.2	General Information	83
6.3	Virtual Cube Server Information	83
6.4	Metadata and Dimension Hierarchies of Virtual Cube	84
6.5	Query	86
6.6	Result	88
7	Conclusion and Future Work	89
A	WebRowSet XML Schema Definition	91
B	OLAP Modelling Markup Language 2.0 XML Schema Definition	94
C	Class Diagrams of the OLAP Administrator Project	97
	Bibliography	103

List of Figures

1.1	Administrator and query client of the OLAP cloud	2
1.2	Data sources for the elastic OLAP cloud	3
1.3	OGSA-DAI as middleware between OLAP administrator and heterogeneous databases	3
1.4	Data integration with OGSA-DAI	4
1.5	OMML between the client system and the elastic OLAP cloud	5
1.6	Step by step development procedure	6
2.1	Hierarchical levels of time dimension	9
2.2	An example of a sales table consisting of three dimensions. [Ona05]	10
2.3	An example of data cube	11
2.4	An example of aggregation query	12
2.5	Dimensions in different hierarchical levels	13
2.6	Roll-up and drill-down on a cube	13
2.7	Slice operation on a data cube	14
2.8	Dice operation on a data cube	15
2.9	Pivot operation between two dimensions	15
2.10	Pivot operation between two different hierarchical levels in dimension Date	16
3.1	Multi-tier Architecture of the client systems	17
3.2	Data flow of the OLAP query client	20
3.3	Data flow of the OLAP administrator	21
3.4	Most packages in the GWT library	22
3.5	Apache Wink High Level Client Architecture Overview [pro10]	25
3.6	Role of the related file: OLAPServiceImpl	25
3.7	Class diagram of OLAPServiceImpl class in the OLAP query client project	26
3.8	Class diagram of OLAPServiceImpl class in the OLAP administrator project	27
3.9	Time cost for initiating virtual cube	28
3.10	Derby network server mode	29
3.11	Role of the DataServiceImpl class in the project	29
3.12	OGSA-DAI architecture	32
3.13	OGSA-DAI workflow	33
3.14	Data service, data service resources and data resources	34
3.15	Role of OgsaDaiServiceImpl class in OLAP administrator project	35

3.16	Workflow visualization of a DQP query	39
3.17	Inheritance structure of JDBC WebRowSet interface	40
3.18	Convert currentRow to an object of Record2 class	44
3.19	GWT run configuration window	46
3.20	Class diagram of the OLAP query client project	52
3.21	Class diagram of the OLAP administrator project	55
4.1	Add GWT plugin repository location	60
4.2	Plugins list from the repository	61
4.3	Confirm the installation	61
4.4	Accept the license agreement	62
4.5	The new Google toolbar	62
4.6	Environment variables	63
4.7	Start Tomcat 6.0	63
4.8	Import project window	64
4.9	Project properties window	64
4.10	Create new user library	65
4.11	Add the required JAR files	65
4.12	Edit library window	66
4.13	GWT Design view	66
4.14	Sysinfo from Derby	67
4.15	Deploy OGSA-DAI Axis onto Tomcat	68
4.16	OGSA-DAI: deployed services list	69
4.17	Directory structure of the OLAP administrator project	70
4.18	Initial appearance of the OLAP query client GUI	71
4.19	Deployed resources on OGSA-DAI server	73
4.20	DQP resource on OGSA-DAI server	74
4.21	Initial appearance of the OLAP administrator GUI	75
5.1	OLAP administrator GUI	78
5.2	Ext GWT grid	79
5.3	OLAP query client GUI	81
C.1	Class diagram of Administrator, Record1 and Record10	98
C.2	Class diagram: OLAPService implementation	99
C.3	Class diagram: DataService implementation	100
C.4	Class diagram: OgsaDaiService implementation	101

List of Tables

3.1	RESTful service interfaces of cloud-enabled OLAP system	24
3.2	Database query result	41
4.1	Applied software tools and libraries	58
4.2	GWT repository locations for other version of Eclipse	60

Chapter 1

Introduction

1.1 Motivation

Cloud Computing [MA09] is a new compute paradigm based on internet. In data center, cloud is made up by thousands of computers and workstations. Hence, Cloud Computing enables more than 10 trillion times of computations per second, with such powerful computing capability Cloud Computing provides a lot of new opportunities for lots of traditional algorithms. Clients can use various kinds of devices such as personal computer, laptop computer, or even a mobile phone to access a cloud data center and perform computing tasks.

On-Line Analytical Processing (OLAP) is an important application of data warehouse technology. OLAP supports complicated analysis on multi-dimensional data, it provides support for decision making with analytical query results which are intuitive and easy to understand. Generally, an OLAP application can organize and integrate the raw data, transform it into multi-dimensional analytical data model, and provides meaningful knowledge out of it to support decision making.

Based on the successful development of the OLAP engine from the GridMiner project [Gri] we decided to design and implement an OLAP system for the Cloud Computing environment. After several months work, the first prototype of the cloud-enabled OLAP system is already available. The prototype system could be deployed in cloud environments with different infrastructure compositions including private cloud, public cloud and hybrid cloud. Performance analysis of the prototype system was done on a public cloud, the Amazon Elastic Compute Cloud (EC2) [Spe06], the result suggests that the system fulfills the requirement of an efficient OLAP system and also meets the characteristics, such as elasticity and virtualization, of Cloud Computing. The prototype system was developed follow the Representational State Transfer (REST) [Fie00] architectural style.

The prototype of cloud-enabled OLAP system brings us an elastic OLAP cloud. For management of and interaction with this elastic OLAP cloud, web based Graphical User Interface (GUI) should be developed. Besides, we should also further develop the OLAP Modelling Markup Language (OMML), which is based on Extensible

Markup Language (XML) [W3C04] and is defined as standard communication language between the OLAP cloud and the clients.

Two aspects are assumed for the management of and interaction with the elastic OLAP cloud. As shown in Figure 1.1 the administrator can read raw data in different format from different data sources, transform it into a uniform format and load it to the OLAP cloud. On the other hand, the client can initiate query, send it to the OLAP cloud and get the analytical result. In this thesis we present and discuss the design and implementation of two multi-tier client systems, one for implementing the administrator's functionality, the other for implementing the query client's functionality.

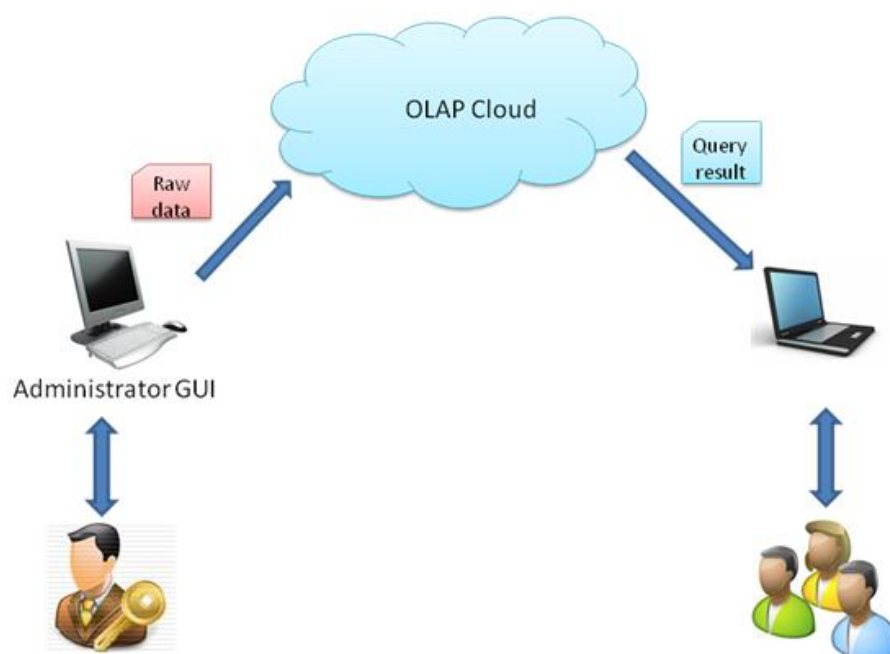


Figure 1.1: Administrator and query client of the OLAP cloud

1.2 Goals

The main focus of the thesis is the development of two multi-tier client systems, one for the administrator and one for query client.

For OLAP administrator system, the main task is to transform the raw data into uniform format and load it to the OLAP cloud. The first thing to consider is: Where the raw data comes from? Simply, relational databases (e.g. Derby database) can be our data sources. Another popular way is that we can get the raw data from an Open Grid Service Architecture - Data Access and Integration (OGSA-DAI) server. The administrator system should be possible to read raw data from both above

mentioned data sources. As shown in Figure 1.2, raw data either from the derby database or from the OGSA-DAI server is to be transformed into WebRowSet XML format before loading to the OLAP cloud.

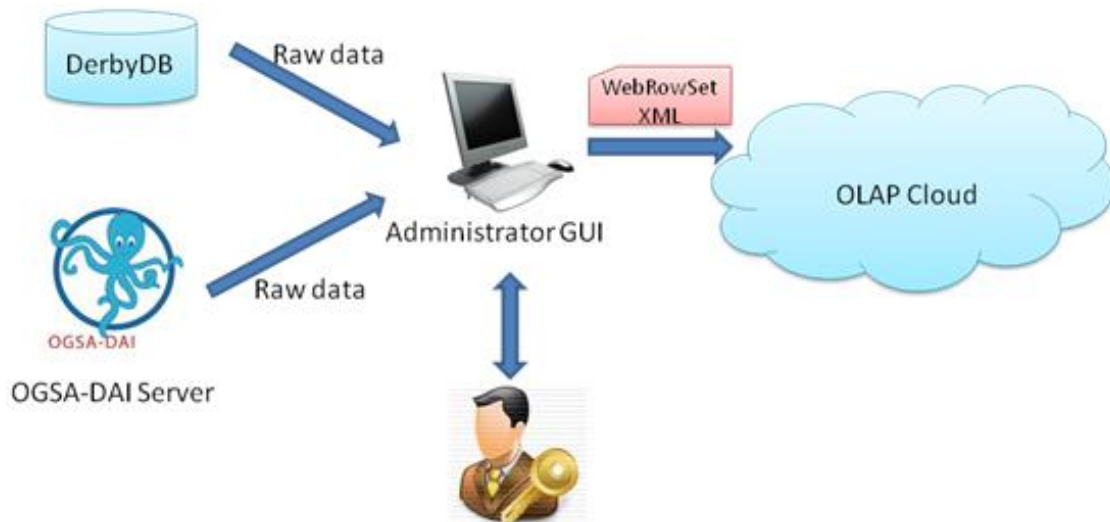


Figure 1.2: Data sources for the elastic OLAP cloud

OGSA-DAI is a middleware which aims to provide an efficient possibility for data access and integration in distributed Grid Computing environment. It allows Grid Computing user and other Grid Computing services to access various kinds of heterogeneous databases including Relational database, XML database and also file system based database. It enables sharing data resources at a high level, so, co-processing and access to the data sources becomes more efficient, transparent and reliable.

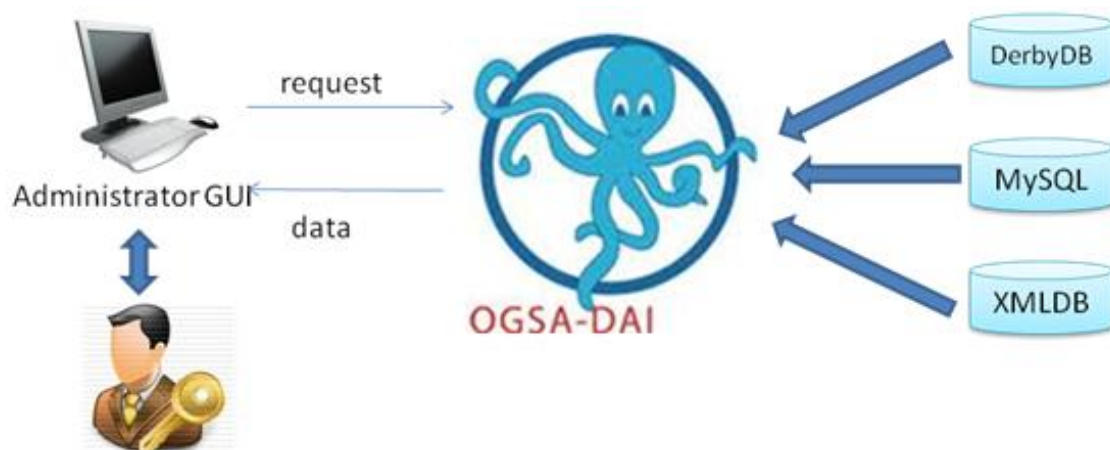


Figure 1.3: OGSA-DAI as middleware between OLAP administrator and heterogeneous databases

As shown in Figure 1.3 in this thesis OGSA-DAI server is a middleware between

OLAP administrator and heterogeneous databases. First, OLAP administrator sends database query to OGSA-DAI server, the OGSA-DAI then queries either a single database or several distributed databases, integrates the results and sends it back to the OLAP administrator.

Figure 1.4 gives an example of the query and data integration workflow of an OGSA-DAI server, which queries multiple databases. When the OLAP administrator's query request arrives at the OGSA-DAI server, the OGSA-DAI server sends at the same time two SQL queries to database 1 and database 2, the two databases execute their own queries respectively, then the results are processed with necessary transformation and integration by the OGSA-DAI server. Finally, the integrated result is send back to OLAP administrator.

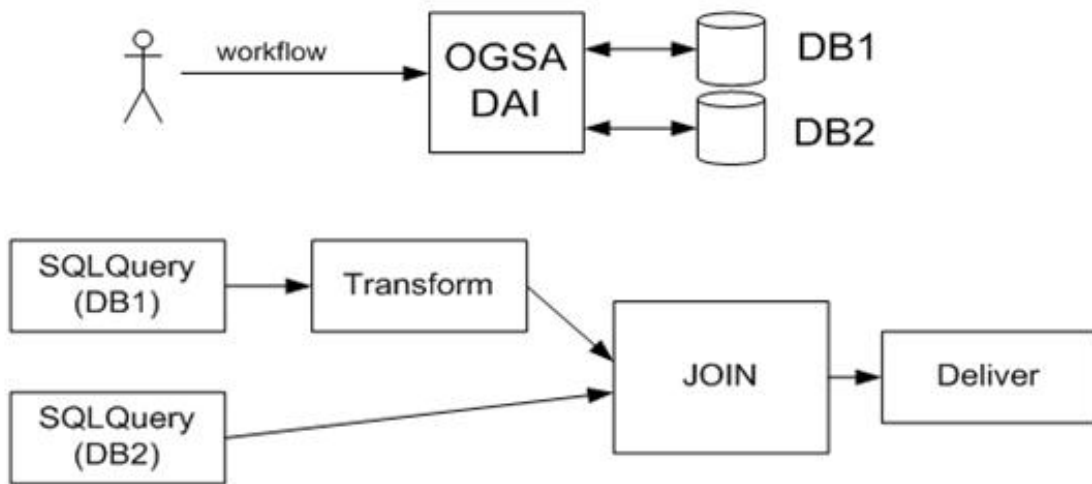


Figure 1.4: Data integration with OGSA-DAI

Another goal of this thesis is to design and implement the OLAP query client system. The main functionality of the OLAP query client includes initiating and sending OLAP queries to the OLAP cloud, receiving results and presenting them in well formatted tables to the user. Here, there is another important task, to further develop and extend the standardized communication language between the elastic OLAP cloud and the client systems, OMML. Figure 1.5 shows where the OMML is used in the system.

The first version of OMML was formulized in [EO05], it was inspired by the Predictive Model Markup Language (PMML) [Gro]. Unlike the PMML which is focus on general predicative models and data mining, the OMML concentrates on presenting the OLAP model and its query results. OMML is designed to provide consistent OLAP model, which can be used by any OMML compatible application. So, besides further develop the OMML standard, the OMML should also be implemented and applied as standard communication language between the elastic OLAP cloud and the client systems.

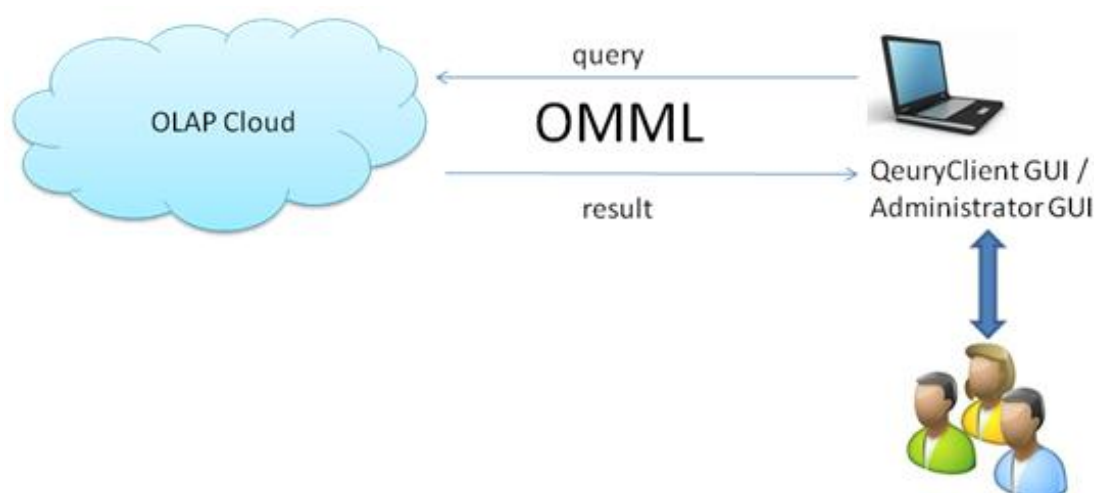


Figure 1.5: OMML between the client system and the elastic OLAP cloud

Google Web Toolkit (GWT) [Goo] is an open source framework based on Java used to fast develop Asynchronous JavaScript and XML (AJAX) [Gar] web application. GWT is distributed by Google as a web application development toolkit, which provides perfect compliance to web 2.0 standard, and can be integrated in many kinds of IDEs like Eclipse and so on. After we deploy our Java coded program to GWT project, GWT compiler will translate the program into JavaScript and html documents that is suitable for various types of browsers.

In this thesis, GWT was mainly used for constructing the two multi-tier client systems' frameworks. Other software tools and libraries including Derby DB, OGSA-DAI, Apache Wink client toolkit, Ext GWT and so on were also applied as third-party library for implementing different aspects of the client systems' functionality.

1.3 Approach

After study the theory of OLAP and comparing different implementation variants of OLAP engines(sequential version, parallel version based on socket communication, parallel version based Java RMI), we decided to first apply the GWT and Apache Wink client module to develop a web based OLAP query client.

The OLAP query client aims at providing an efficient way for the users to send different kinds of queries to the elastic OLAP cloud and receive results, which are well presented in forms to the users. Besides, the task includes also design and implementing OMML for standardizing the communication language between the clients and the elastic OLAP cloud.

During the development of the OLAP query client, we started to think about the

second client system - OLAP administrator. OLAP administrator was also developed using the GWT, besides, libraries of Apache Wink, Apache Derby database and OGSA-DAI, Ext GWT were applied as third-party library and plugin for implementing some business logic.

OLAP administrator client is designed to be used by user with administrator privilege, it inherits some functions from the OLAP query client, moreover, it can also query the Derby database and OGSA-DAI server to load raw data and transform it into WebRowSet format and transfer it to the elastic OLAP cloud.

The exact scheme of the step-by-step development procedure is illustrated in Figure 1.6. Of course, this scheme does not represent our whole software engineering approach, as it does not include all the phases and our development do not need to have been continued during these development phases. This scheme just represents a better step by step procedure.

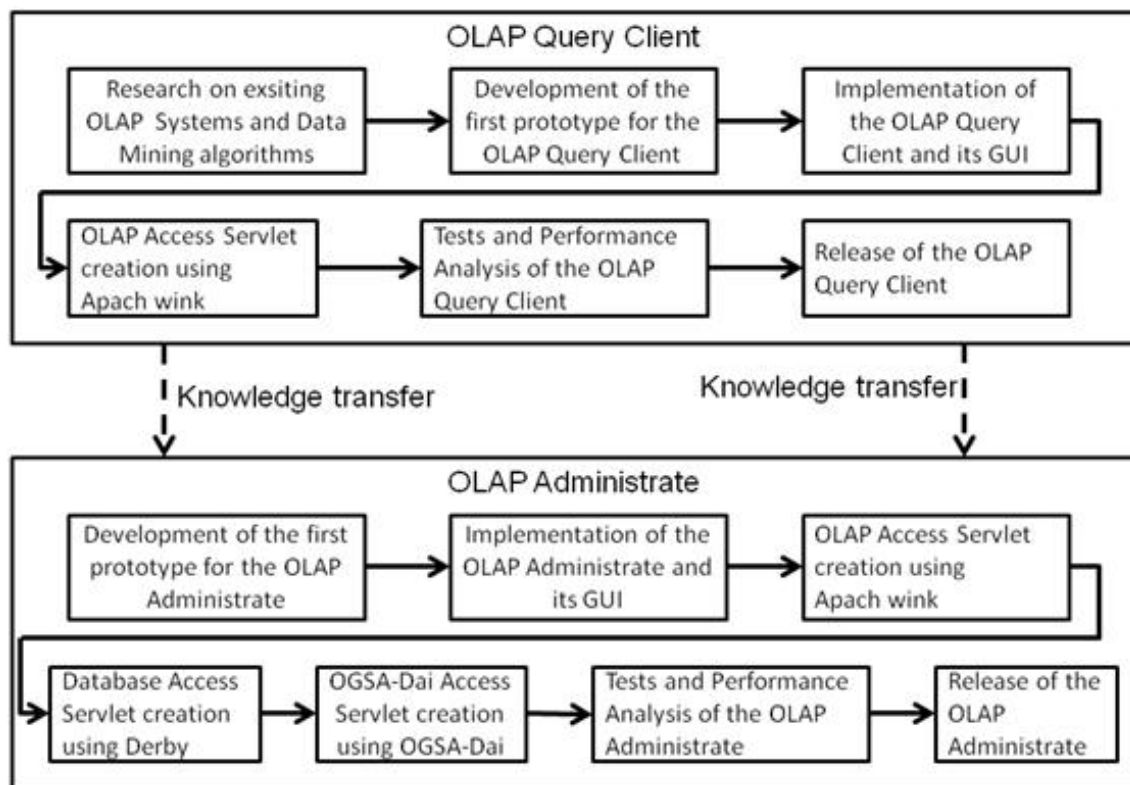


Figure 1.6: Step by step development procedure

1.4 Thesis Organization

In the next chapter there is an introduction to the basics of OLAP including several kinds of OLAP operations. Chapter 3 is detailed description for the system design

and implementation, there are descriptions about the multi-tier architecture, data flow chart, details on the applied software tools and libraries and functionality of each part of the program. Chapter 4 is the guide for installation and deployment of the system which can be the basis for further development and extension. In Chapter 5 we will discuss the details of the OMML version 2.0 schema. In Chapter 6, there is GUI introduction for the both client systems. Finally, conclusion and future work are covered in Chapter 7.

Chapter 2

Basics of On-Line Analytical Processing

Relational model was first formulated and proposed by E.F.Codd in 1969, it encouraged the development of On-Line Transaction Processing (OLTP). In 1993, E.F.Codd proposed the principle of On-Line Analytical Processing (OLAP), as in that time OLTP was already no more sufficient for the requirement of query and analysis for databases, decision making strategy should be supported by intensive analytical computations, but the simple SQL query for huge data warehouse could not fulfill the analytical requirements. So, the idea of multi-dimensional database and multi-dimensional data model was introduced. Nowadays, OLAP becomes an important approach for knowledge discovery.

2.1 Basic Principles

OLAP aims at data access and on-line analysis for specific problem. OLAP enables decision makers for observing information in many possible views, accessing data in an efficient, interactive and high consistent way and perform various kinds of analyses. Following are some basic definitions of OLAP.

Dimension:

A dimension represents a specific aspect of data. For example, enterprises would normally like to observe changes of their product sales over time, i.e. they observe the product sales data from the time aspect, and time is a dimension. If the product sales are observed from the aspect of geographical distribution, then the geographical distribution is a dimension. In both cases the product sales are the measures.

Hierarchy of dimensions:

Based on different level of detail a dimension can have multiple hierarchical levels. For example, a time dimension can be described in different levels like date, month, season and year, so these are the hierarchical levels of the time dimension,

as shown in Figure 2.1. Similarly, district, city and country are the different hierarchical levels of geographical location dimension.

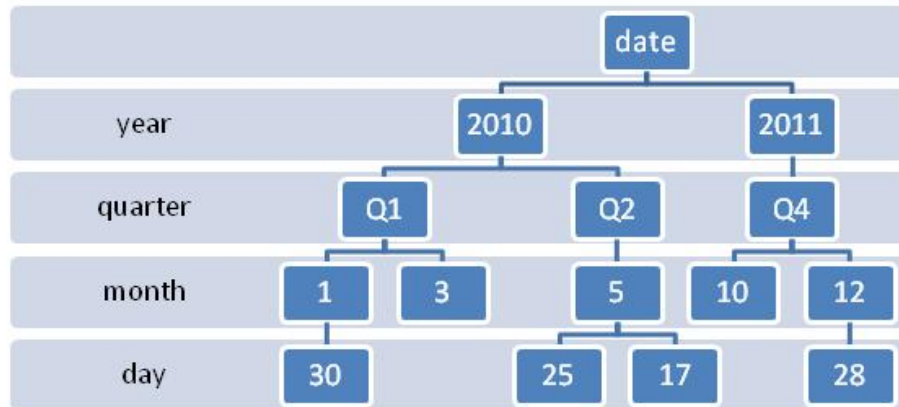


Figure 2.1: Hierarchical levels of time dimension

Dimension members:

A possible value of a dimension is a member of the dimension. In case a dimension has more hierarchical level, then a dimension member is made up by combination of possible values from all its different hierarchical levels. For instance in Figure 2.1, a dimension member of time dimension can be *(28th, December, 4th_quarter, 2011)*.

Multi-dimensional Array:

A multi-dimensional dataset can be represented using such a multi-dimensional array: $(dimension1, dimension2, dimension3, measure)$. For example, product sales data is a three-dimensional array made up by time, location, product and measure (product sales): $(location, time, product, sales)$. Such a combination of dimensions and measure is given in Figure 2.2.

Cube:

The three-dimensional array in Figure 2.2 can be represented using a data cube. The relationship of dimensions, hierarchical levels and the measures can be represented by a cube as shown in Figure 2.3. But a cube must not be limited to have three dimensions, normally a multi-dimensional data array can be represented by multi-dimensional cube, so it is also called hypercube.

Location	Date	Product	Sales
USA	11/12/1999	TV	120
USA	06/02/2000	Radio	158
USA	06/01/2001	Radio	91
USA	06/02/2001	TV	91
USA	06/03/2001	Radio	91
Denmark	11/12/1999	DVD Player	45
Denmark	06/02/2000	DVD Player	78
Denmark	06/01/2001	DVD Player	120
Denmark	06/02/2001	TV	35
Denmark	06/03/2001	TV	35
Austria	11/12/1999	TV	21
Austria	06/02/2000	TV	35
Austria	06/01/2001	Radio	155
Austria	06/02/2001	Radio	16
Austria	06/03/2001	TV	21
Austria	06/03/2001	DVD Player	35
Austria	06/03/2001	Radio	155
Japan	11/12/1999	TV	45
Japan	06/02/2000	DVD Player	65
Japan	06/01/2001	TV	72
Japan	06/02/2001	TV	45
Japan	06/03/2001	DVD Player	65
Japan	06/03/2001	TV	72
Japan	06/03/2001	Radio	45

Figure 2.2: An example of a sales table consisting of three dimensions. [Ona05]

A hypercube structure means that use three or more dimensions to describe a object, each dimension is orthogonal to each other. Measured value of data is occurred in the intersection point of the dimensions.

This structure can be applied in multi-dimensional database and OLAP system oriented for RDBMS, its main feature is that it simplifies the operation of end-user.

The approach of OLAP cube enables efficient data access and also simplifies the aggregation computation especially for huge amount of data. Aggregation computations like counting, sum, average and so on need always to repeat again and again, in OLAP cube, the aggregation computation results can be cached to guarantee rapid response to different aggregation queries.

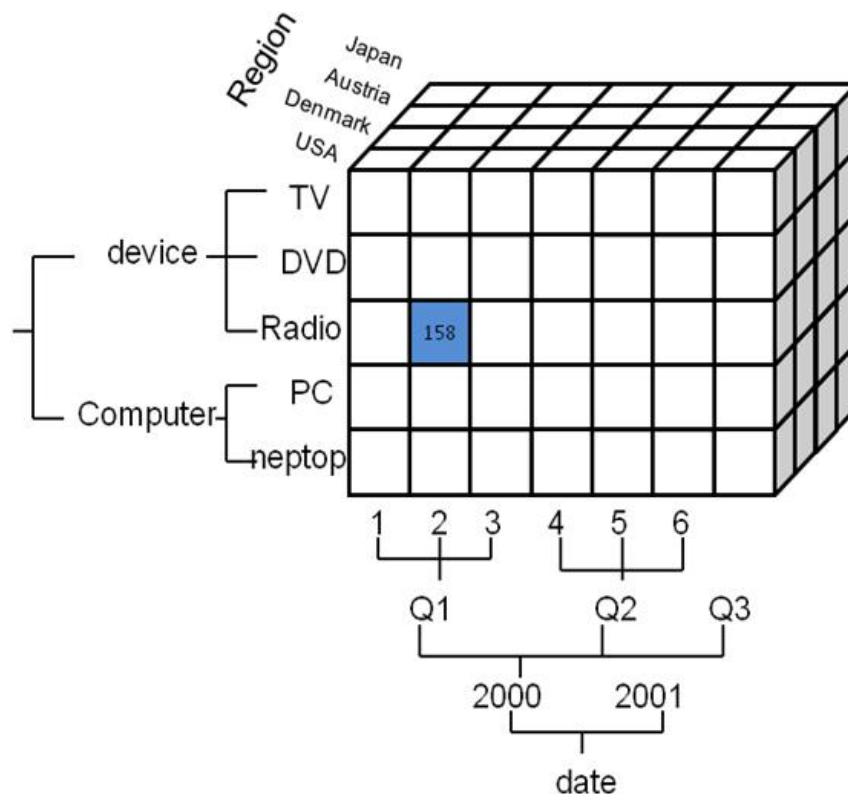


Figure 2.3: An example of data cube

2.2 Classification of OLAP

According to the data storage format, OLAP systems can be classified into three major types.

Relational OLAP (ROLAP)

ROLAP represents the OLAP implementation based on relational database, it uses relational data model to describe the multi-dimensional data. There two kinds of tables in ROLAP for multi-dimensional data: The first is fact table, which is used to store data and dimension keywords. Another is dimension table, which uses at least one table for each dimension for storing the hierarchy information, dimension member type and other description about the dimension. Fact tables and Dimension tables are connected together via keywords to form a "star schema". Dimensions with complicated hierarchy can be described by multiple tables, which connected together following a more complicated "snowflake schema".

Muilt-dimensional OLAP (MOLAP)

MOLAP is implemented on basis of multi-dimensional data cube model. For MOLAP, there are many possible operations, which can be performed on cube, these

includes pivot, slice, dice, roll-up and drill-down. Different views of data can be generated by applying these operations. The OLAP engine from GridMiner project was also implemented as MOLAP, detailed description of this OLAP engine can be found in [FB04b] and [FB04a].

Hybrid OLAP (HOLAP)

MOLAP and ROLAP has each respective advantage and disadvantage, and their data structures are totally different. In order to integrate their advantages in a uniform model, HOLAP was proposed as a combination of MOLAP and ROLAP. Data in HOLAP is organized in a hybrid way, for example relational model in low level and multi-dimensional model in high level for better flexibility.

2.3 MOLAP Operations

As our elastic OLAP cloud was implemented based on the GridMiner's OLAP engine, which was implemented as MOLAP, in this section we introduce some OLAP operations and describe them in context of MOLAP.

2.3.1 Aggregation

Aggregation is the most commonly applied OLAP operation, it computes the aggregated result by applying different operation methods (Aggregators) like *SUM*, *MIN*, *MAX* or *AVERAGE* along one or multiple dimensions of a cube. Aggregation is the basis for other operations. Figure 2.4 gives an example where dimension Date is set to be *ANY*, other dimensions are set to be a specific dimension member and sales is the measure. So, if we apply the aggregator *SUM*, this aggregation operation will calculate the sum of PC sales in Wien at any time.

Date	Location	Product	Sale
Any	Wien	PC	?

Figure 2.4: An example of aggregation query

2.3.2 Roll-up and Drill-down

As mentioned before, a dimension could be described in different hierarchical levels. High hierarchical level means more abstract, summarized data and so relatively fewer amount of data. Low hierarchical level means more detailed data and so relatively larger amount of data. By applying drill-down operation we can observe

abstract data at a detailed level, and roll-up is the operation to observe detailed data at an abstract level. Let's introduce an example as shown in Figure 2.5, it presents the product sales analysis using three dimensions: Product, Date and Location. For example, the dimension Product can be drilled-down into more detailed level as indicated by the bold line: Product, Device, DVD.

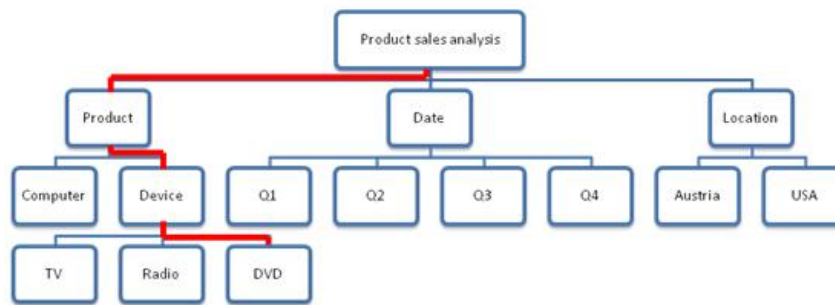


Figure 2.5: Dimensions in different hierarchical levels

Based on the above example, Figure 2.6 uses MOLAP cube to represent drill-down and roll-up operation. On the left side, the dimension Date can be drilled-down at point Q1 into month. On the other hand, the dimension Location can be rolled-up into a more abstract dimension level country.

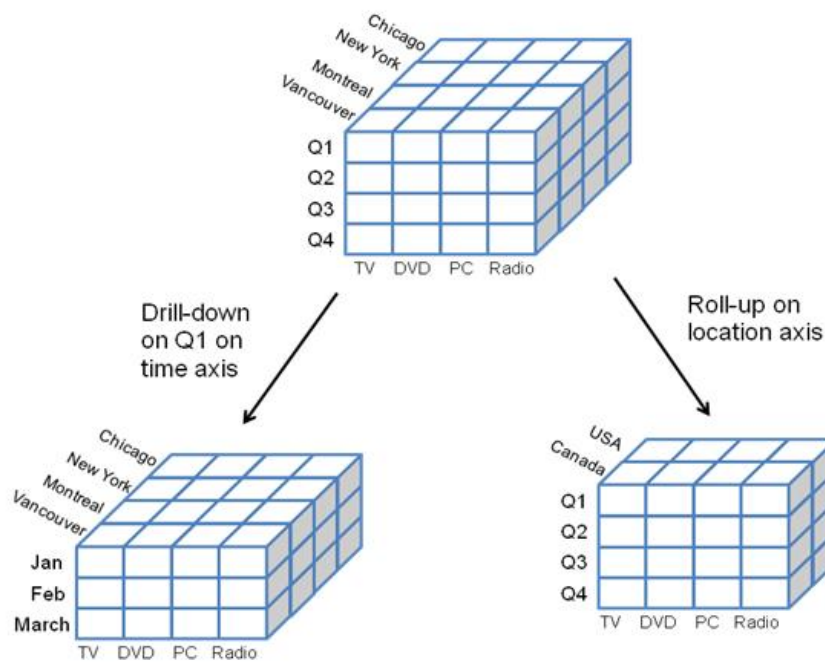


Figure 2.6: Roll-up and drill-down on a cube

2.3.3 Slice and Dice

Slice

Definition 1: If a specific value is fixed for a dimension, then the N -dimensional data is down to $(N - 1)$ -dimensional data. The combination $(dimension1, dimension2, \dots, dimensionmemberV_i, dimensionN, measure)$ represents the slice on dimension i . Definition 2 is more general.

Definition 2: Specify *ANY* or an interval for two dimensions, dimension i and dimension j , assign other dimensions each a specific dimension member, then the result is called slice on dimension i and dimension j .

For example, in Figure 2.7, the cubes describe the product sales. The shadow part of the left cube represents the slice operation for dimension Product is equal *TV*. The shadow part of the right cube represents the slice operation for dimension Date is equal *Q1*.

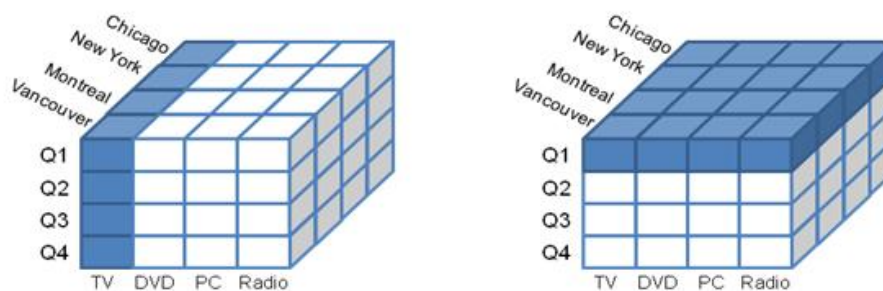


Figure 2.7: Slice operation on a data cube

Dice

Definition 1: If we set an interval for a dimension (e.g. 2005 - 2011 for dimension date), and set all other dimensions to be *ANY*, this operation is called dice. When the interval has only a single value, then the operation is also slice.

Definition 2: The operation of restricting intervals for one or multiple dimensions (e.g. dimension i , dimension j , ...), and assigning for the rest dimensions each a specific dimension member is called dice on these dimensions (dimension i , dimension j , ...).

In Figure 2.8, the shadow part of the left cube represents the dice operation on dimension location by giving the interval $[NewYork, Montreal]$. The shadow part of the right cube represents the dice operation on all dimensions of the cube by giving interval $[Montreal, Vancouver]$ for dimension Location, giving interval $[Q1, Q2]$ for dimension Date and giving interval $[PC, Radio]$ for dimension Product.

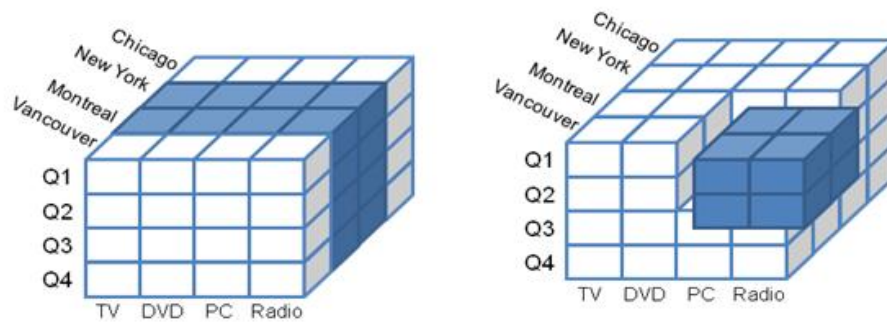


Figure 2.8: Dice operation on a data cube

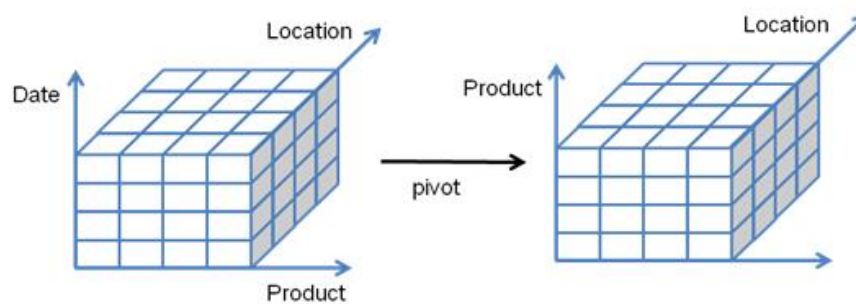


Figure 2.9: Pivot operation between two dimensions

2.3.4 Pivot

The operation pivot enables generation of different data views by exchanging roles between dimensions. Pivot can be performed either between dimensions like the exchange of dimension Date and dimension Product as shown in Figure 2.9, or between different hierarchical levels of a dimension like the exchange of level year and level quarter as shown in Figure 2.10.

	2000				2001			
Product	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
TV	200	150	128	312	129	209	140	270
Radio	157	265	309	104	288	305	246	312
DVD	220	176	289	299	190	387	150	274

↓ pivot

	Q1		Q2		Q3		Q4	
Product	2000	2001	2000	2001	2000	2001	2000	2001
TV	200	129	150	209	128	140	312	270
Radio	157	288	265	305	309	246	104	312
DVD	220	190	176	387	289	150	299	274

Figure 2.10: Pivot operation between two different hierarchical levels in dimension Date

Chapter 3

Design and Implementation

3.1 Multi-tier Architecture

Figure 3.1 presents the multi-tier architecture for both OLAP query client and OLAP administrator. Although the two client systems were implemented as individual projects, the multi-tier architecture is given here just to show the same main structure of the two systems, the internal implementation of respective functional modules are not the same.

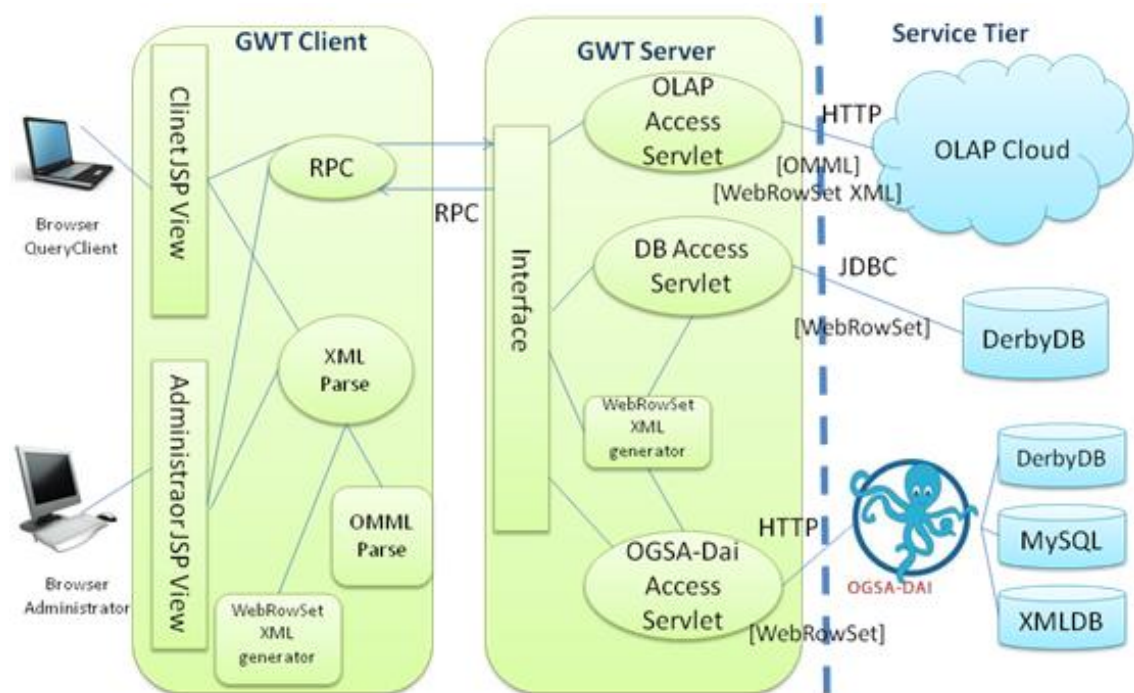


Figure 3.1: Multi-tier Architecture of the client systems

In Figure 3.1, the part left to the dashed line is the main programming part. In the programming part there are two tiers, one is the GWT client tier which was developed mainly using GWT, another is the GWT server tier which was developed based on GWT and also using OGSA-DAI, Apache Wink and Derby as third-party library.

The right part of the dashed line there are three services in communication with the client systems. For this part, installation and configuration is needed, this part is the service tier.

3.1.1 Design of the Service Tier

The service tier contains following three parts:

1. **OLAP cloud:** The OLAP cloud provides OLAP services to the client systems. We just need to concentrate on the RESTful interfaces of the OLAP cloud, but do not need to consider the internal details of it.
2. **Derby Database:** As the administrator system is designed to get raw data from Apache Derby database, so here we should install and configure the Derby database server and create databases and tables for data used in experiments.
3. **OGSA-DAI server:** The administrator system can also get raw data from OGSA-DAI server. We should install and configure OGSA-DAI server and deploy Derby database as data resources and also as DQP resources to it.

3.1.2 Design of the GWT Client Tier

The main tasks of the GWT client tier:

1. **JSP views:** Implement two different JavaServer Pages (JSP) [Ora] views, one for OLAP query client and one for OLAP administrator.
2. **RPC:** Implement Remote Procedure Call (RPC) communication between GWT client tier and GWT server tier.
3. **OMML Parser:** Implement the function for converting the query table from user to OMML document and converting the OMML query result from the OLAP cloud and present it to user as table.
4. **WebRowSet XML Parser:** First, in the GWT server tier the raw data from the Derby database or from OGSA-DAI server will be transformed into the uniform WebRowSet XML format. Then the data will be passed to the GWT client tier, in the GWT client tier the WebRowSet XML format data is parsed and presented in grid to user, the user can then decide if the data shown in the table should be loaded to the OLAP cloud.

3.1.3 Design of the GWT Server Tier

Business logic of the GWT server:

1. **OLAP access servlet:** This servlet communicate with the OLAP cloud through HTTP and implemented by Apache Wink client toolkit. The servlet handles the request to send to the OLAP cloud, transferring parameters and getting response. For loading data to the OLAP cloud, the servlet transfers data in WebRowSet XML format to OLAP cloud. For querying OLAP, the servlet communicates with the OLAP cloud by exchanging OMML information.
2. **Database access servlet:** This servlet implements data access to Derby database. It uses JDBC to connect to Derby databases and get data from the database and transform the raw data into WebRowSet XML format and sends it to GWT client tier.
3. **OGSA-DAI access servlet:** This servlet implements data access to OGSA-DAI server. OGSA-DAI server integrates data from heterogeneous data sources and provide it to the servlet. The servlet transforms the raw data into WebRowSet XML format and sends it GWT client tier.

The three servlets from above implement the main business logic of the GWT server tier. There is a WebRowSet XML generator functional module which is embedded both in database access servlet and in OGSA-DAI access servlet, its main function is converting the WebRowSet Java object, which holds the resulted data, to WebRowSet XML.

3.2 Data Flow in the System

Next, we introduce the system from the aspect of data flow. As the two client systems were implemented as individual projects. Their data flow will be described respectively.

3.2.1 Data Flow of the OLAP Query Client

Figure 3.2 shows the data flow diagram of the OLAP query client project. Firstly, the user should create query form in the GUI and fill the form with query parameters, the form will be then parsed by the OMML parser module and converted to OMML message. After that, a RPC connection will be established between the GWT client tier and GWT server tier, the OMML message will be transferred to the GWT server tier through the RPC connection and then will be sent to the OLAP cloud by OLAP access servlet through HTTP.

In the service tier, the OLAP cloud will handle the query sent by the OLAP query client. After the result is available in OMML format, it will be transferred back through the above mentioned two connections: through HTTP to the GWT server tier and then through RPC connection to the GWT client tier. Finally, in the GWT client tier, the query result in OMML format will be parsed again by the OMML

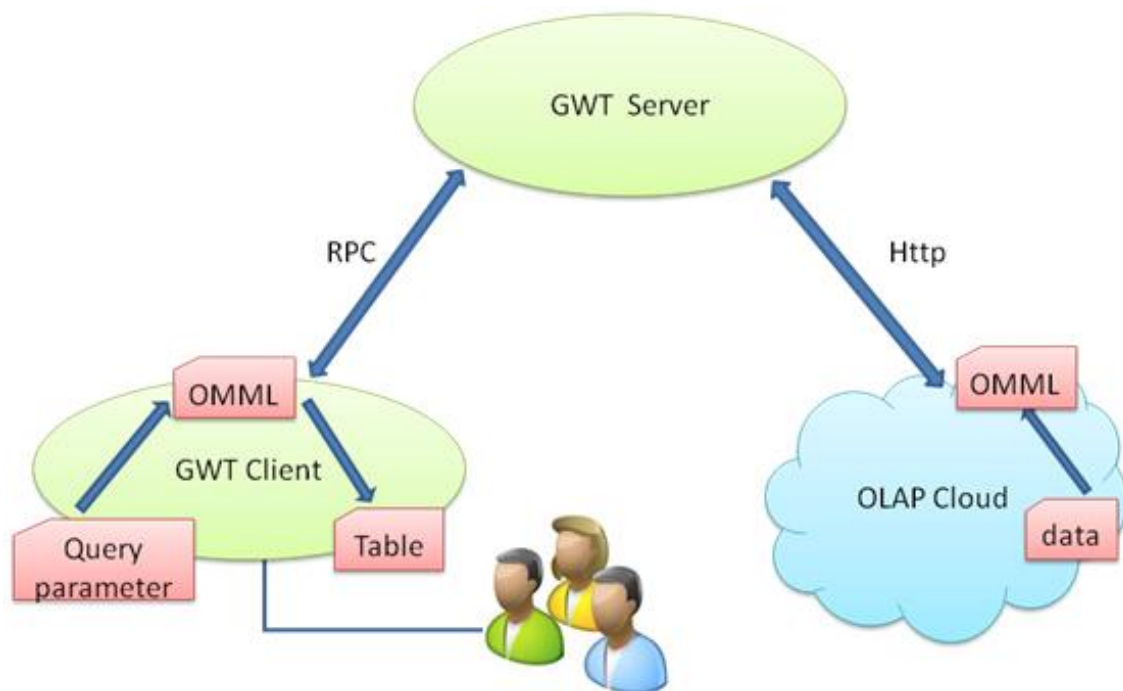


Figure 3.2: Data flow of the OLAP query client

parser module and will be presented to the user as table in the GUI.

3.2.2 Data Flow of the OLAP Administrator

Figure 3.3 presents the data flow diagram of the OLAP administrator project in the multi-tier architecture. Firstly, the user as administrator can initiate request for raw data in the OLAP administrator GUI, the request will be sent to the GWT server tier through RPC connection. After that, based on the user's specification, if the requested raw data is provided by a Derby database directly, the database access servlet in the GWT server tier will establish a JDBC connection to the Derby database and forward the request through the connection. If the requested raw data is provided by OGSA-DAI server, the OGSA-DAI access servlet will forward the request through HTTP to the OGSA-DAI server. After the requested raw data is sent back to the GWT server tier, it will be transformed into WebRowSet XML format before return to the GWT client tier.

In the GWT client tier, the returned WebRowSet XML data will be parsed by the WebRowSet XML parser module and will be transformed into grid to present to the administrator user in the GUI. The administrator user will then decide if the data is to be loaded to the OLAP cloud, or if the data is not proper, the user can also make some changes on the data request and get raw data again from database or OGSA-DAI server.

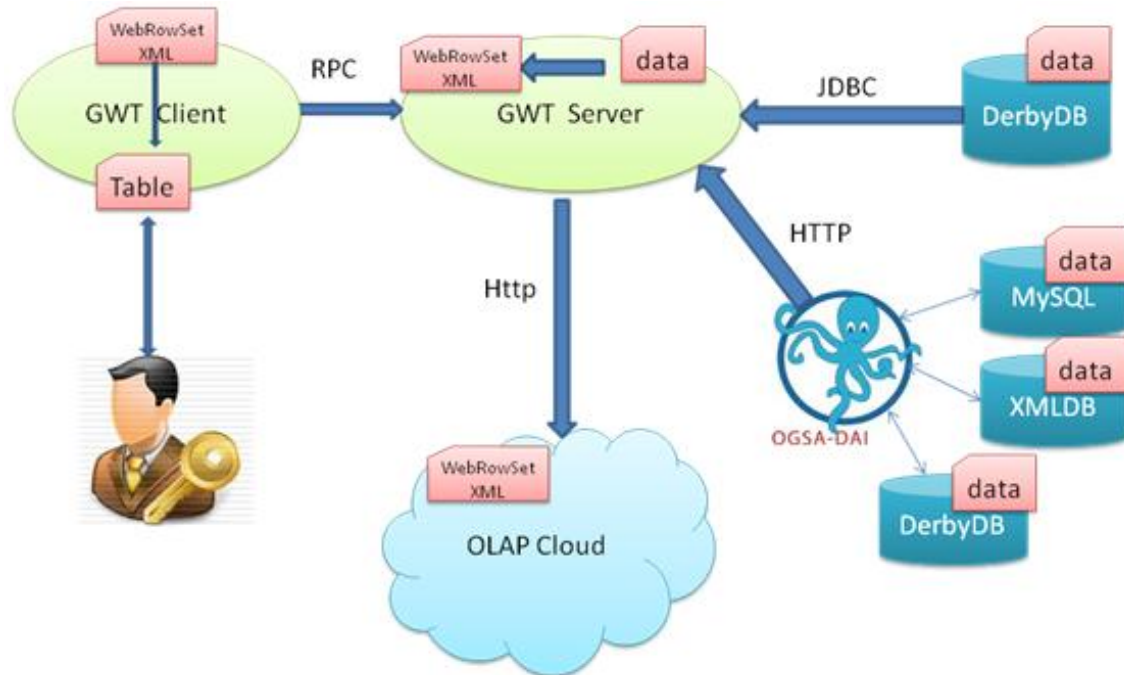


Figure 3.3: Data flow of the OLAP administrator

3.3 Why Multi-tier Architecture

Actually, there is simpler way to access the OLAP cloud, as we can directly send HTTP request from the GWT client tier to the OLAP cloud. Why we design such a multi-tier architecture? And why the GWT server tier is significant in the system? To answer the two questions, we should firstly introduce the applicability of libraries, packages and GWT plugins in the GWT application development.

GWT library

The main advantage of the GWT development framework is that we can write our program in Java and fast develop Web 2.0 compliant web application. GWT uses the file gwt-user.jar to encapsulate the essential packages of the GWT, the function of the library can be categorized into following groups: user interface, server calls, data formats, JRE and utility. (Figure 3.4)

Third-party libraries and plugins

As described in the previous section, there are only limited packages which can facilitate our development. For extending functionality of our application, we should also apply third-party libraries and plugins, these can be classified into following two groups.

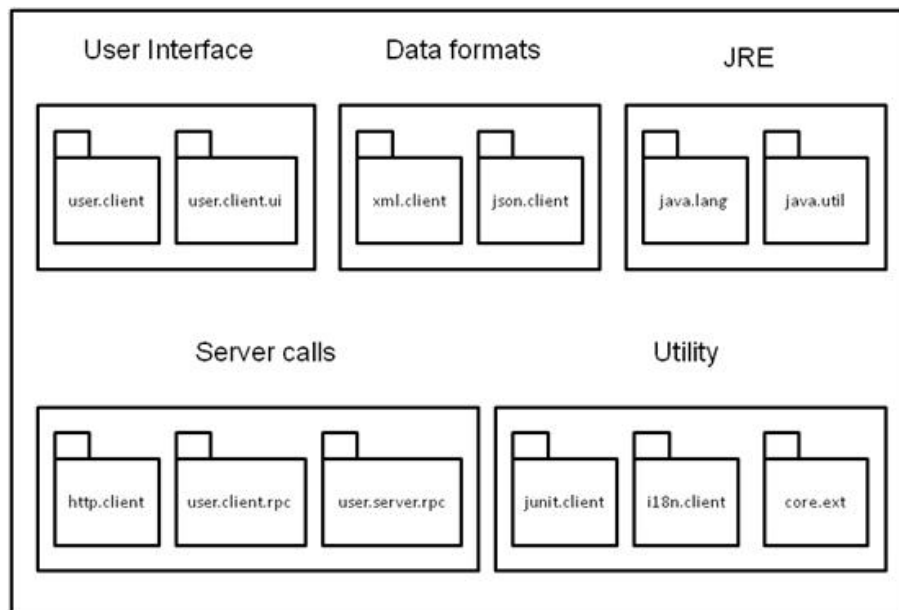


Figure 3.4: Most packages in the GWT library

1. Dedicated for GWT

There are a series of plugins which can be applied specifically for GWT development. These plugins can be easily configured and applied in the development of GWT client tier. Popular GWT plugins include MYgwt, GWT Ext, Ext GWT, hibernate4gwt and so on.

In the OLAP administrator project, Ext GWT was applied. It is a powerful software component library, it provides data grid which can be sorted, paged and filtered, it also provides draggable tree, tab panels, menus, toolbars, dialogs, forms and other attractive GUI components.

2. Third-party libraries for plain Java program

Not most popular Java libraries can be applied in GWT client tier development. For example, XStream [XSt] is a useful library for handling XML documents, but it cannot be simply used in GWT client tier development. Because the client tier code and all its used libraries will be compiled to JSP web application, if we apply some third-party library in the client tier, the library will often cause exception or failure because of compatibility or other unexpected errors.

Luckily, we can apply third-party Java libraries in the GWT server tier development, the applied libraries in the server tier will be compiled just like normal Java servlet application. In the projects, libraries of Apache Wink, Derby database and OGSA-DAI were applied in the GWT server tier, and they cannot be applied in GWT client tier, that is the reason why we designed such a multi-tier architecture for the client systems.

3.4 Implementation

3.4.1 Implementation of OLAP Access Servlet

Main function: Implement the communication between GWT server tier and the OLAP cloud.

Applied tool: Apache Wink client toolkit

Related class files: OLAPServiceImpl.java in the OLAP query client project and OLAPServiceImpl.java in the OLAP administrator project.

As mentioned before, the OLAP access servlet helps us to send HTTP request and handle the response so to interact with the OLAP cloud's RESTful interfaces which were implemented using the Apache Wink server module. So, it is a good choice for us to apply the Apache Wink client module to implement the OLAP access servlet to guarantee high compatibility.

Overview

The Apache Wink client toolkit provides us a high level Java API for writing clients that consume HTTP-based RESTful Web Services. The Apache Wink client toolkit follows JAX-RS standard and encapsulates REST standards and protocols, it maps REST concepts to Java classes which facilitate the development at the client side for HTTP-based REST Web Services. Besides, it also provides a handlers mechanism to enable the manipulation of various kinds of HTTP request and response messages.

The RestClient class is the entry point of the Apache Wink client toolkit, it should be instantiated as an object before we can use it. Figure 3.5 illustrates the principle structure of the RestClient.

First, resource objects should be generated by RestClient. A resource object represents a web service resource related to a certain Unified Resource Identifier (URI). E.g. in the OLAP cloud, a virtual cube is a resource which related to the URI: `http://{serverIp}:{port}/VCubes/{VirtualCubeID}`. Next, RestClient can perform standard HTTP methods, including GET, PUT, POST and DELETE, on the resource objects. Request will be sent to the resource object related URI, and the response will be handled.

REST API of the OLAP cloud

Table 3.1 lists the RESTful service interfaces provided by the elastic OLAP cloud.

Invoke Method	URI	functionality
GET	http://{serverIP}:{port}/VCubes	Get general information about all available virtual cubes on a server
GET	http://{serverIP}:{port}/VCubes/{VCubeID}	Get metadata of a specific virtual cube
POST	http://{serverIP}:{port}/VCubes	Initiate a new virtual cube on the server
POST	http://{serverIP}:{port}/VCubes/{VCubeID}/aggregate	Perform an OLAP Query on a virtual cube
DELETE	http://{serverIP}:{port}/VCubes/{VCubeID}	Delete a virtual cube
POST	http://{serverIP}:{port}/VCubes/{VCubeID}/loadData	Load data to a virtual cube
GET	http://{serverIP}:{port}/VCubes/hostpool	Get the list of registered hosts on a virtual cube server
GET	http://{serverIP}:{port}/VCubes/hostpool/register	Register a host to the virtual cube server
GET	http://{serverIP}:{port}/VCubes/hostpool/deregister	Deregister a host from the virtual cube server
POST	http://{serverIP}:{port}/VCubes/{VCubeID}/loadOneRow	Load one row of data record to the virtual cube
POST	http://{serverIP}:{port}/VCubes/hostBroker	Set the virtual cube server's host broker
POST	http://{serverIP}:{port}/VCubes/hostBroker	Show service URI of the virtual cube server's host broker

Table 3.1: RESTful service interfaces of cloud-enabled OLAP system

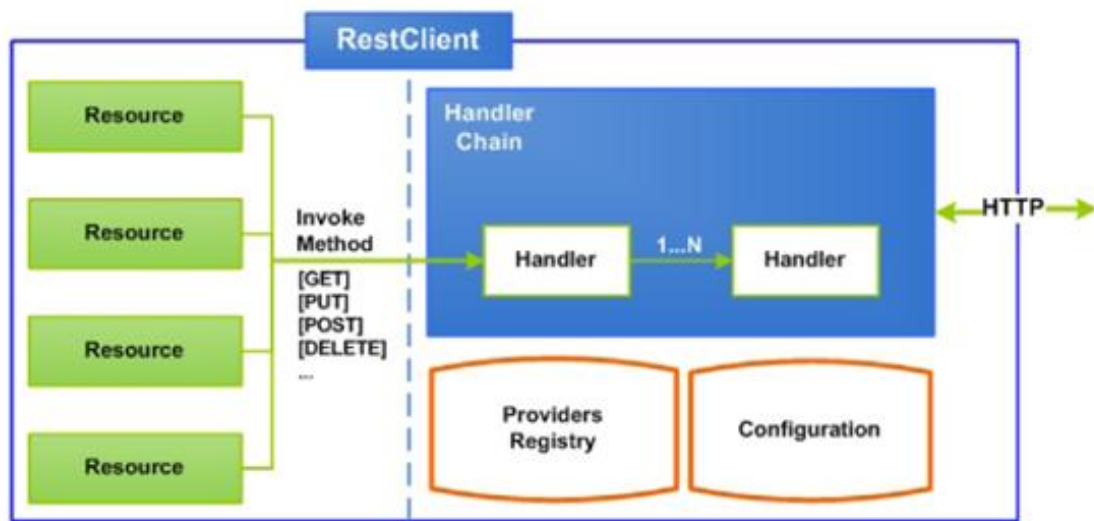


Figure 3.5: Apache Wink High Level Client Architecture Overview [pro10]

Role of the related file

Figure 3.6 presents the role of the `OLAPServiceImpl` class, which is in the package `com.google.gwt.query.server` in the GWT server tier. On one hand, the `OLAPServiceImpl` class implements the RPC interface for the RPC call from the GWT client tier, on the other hand, it works as a client that consume RESTful web services of the OLAP cloud.



Figure 3.6: Role of the related file: `OLAPServiceImpl`

Implementation detail of the `OLAPServiceImpl` class in the OLAP query client project

As shown in Figure 3.7, there are three methods implemented by the `OLAPServiceImpl` class in the OLAP query client project:

doGet(): The method is used to handle the HTTP GET request. E.g. it is used to get the metadata of a virtual cube or get the server information of a virtual cube server.

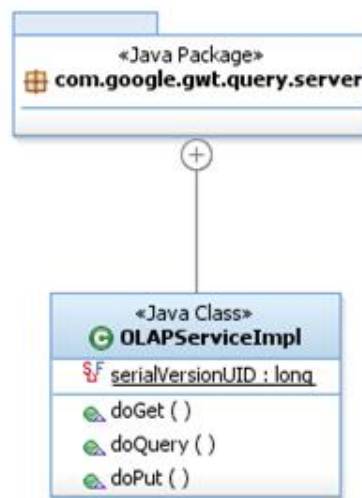


Figure 3.7: Class diagram of OLAPServiceImpl class in the OLAP query client project

1. The method create an object of the RestClient class.
2. By calling the `RestClient.resource()` method, a resource, which relates to a certain service URI, is created.
3. Finally, the `Resource.get()` method is called, a HTTP GET request is sent. Once the Http response is returned, the client invokes the relevant provider to desterializes the response.

doQuery(): This method is mainly used to handle the HTTP POST request. It is used to send OLAP query to a virtual cube in the OLAP cloud.

1. Like the GET request, an object of the RestClient class is generated.
2. By calling the `RestClient.resource()` method, a resource, which relates to a certain service URI, is created.
3. Finally, the `Resource.post()` method is called, a HTTP POST request is sent. Here the content type and the accept type of the HTTP POST request should be specified as XML format, as we use OMML to exchange massages.

doPut(): The method is used to load one row of data to a virtual cube in the OLAP cloud. Except the service URI is not the same, this method is similar to the `doQuery()` method, it also handles HTTP POST request. Besides, the accept type is not XML but plain text.

Implementation detail of the OLAPServiceImpl class in the OLAP administrator project

As shown in Figure 3.8, there are four methods implemented by the OLAPServiceImpl class in the OLAP administrator project:

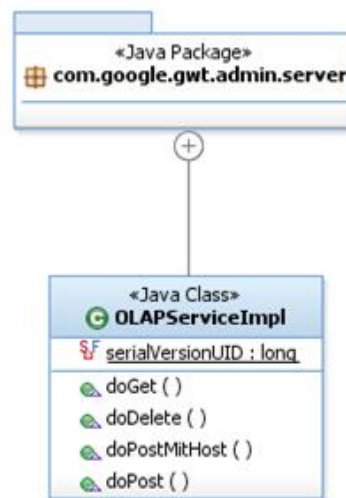


Figure 3.8: Class diagram of OLAPServiceImpl class in the OLAP administrator project

doGet(): The method is used to handle the HTTP GET request just like the doGet method in the OLAP query client project.

doDelete(): This method handles HTTP DELETE request, it is used to delete virtual cube in the OLAP cloud.

1. A RestClient object is generated.
2. By calling the RestClient.resource() method, a resource is created.
3. Finally, Resource.delete() method is called to send a HTTP delete request to the service resource, and the accept type of the response should be plain text.

doPostMitHost(): The method handles HTTP POST request, and it is used to initiate new virtual cube in the OLAP cloud. additionally, a ClientConfig class should be used to configure the timeout setting of the client.

1. An object of the ClientConfig class is generated.
2. The property connectTimeout and the property readTimeout of the object is set to 500,000, which means the timeout is 500 seconds.
3. Then the ClientConfig object is used as parameter in the constructor method of the RestClient class to create an RestClient object.
4. In the HTTP POST request the content should be set as the metadata of the new virtual cube in OMML format and the accept type is plain text.

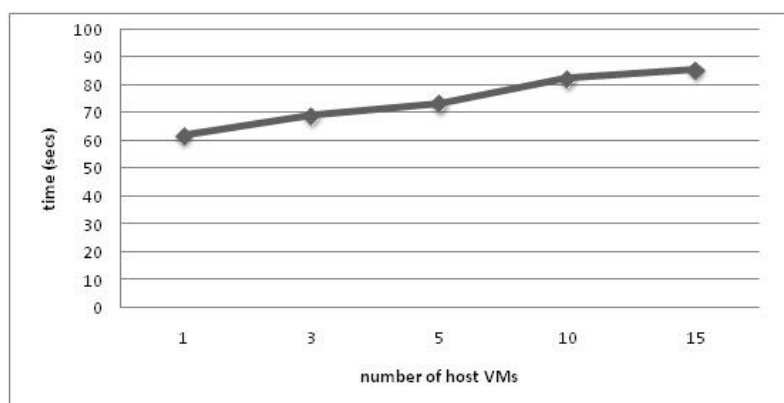


Figure 3.9: Time cost for initiating virtual cube

As mentioned above, the timeout of the client is set to be 500 seconds, this is because that the method is used to initiate new virtual cube in the OLAP cloud, and this is the most costly procedure in the system. The method interact with the service URI `http://{serverIp}:{port}/VCubes?nHosts`, the parameter `nHosts` indicates how many hosts are used by the new virtual cube. As a result, greater number of `nHost` means more time cost for initiating the new virtual cube. Figure 3.9 shows the relationship between the number of hosts and time cost for initiating new virtual cube. For a single host, we need to wait about 62 seconds, and for fifteen hosts about 86 seconds is needed. So, a relative long timeout should be set and we set it to be six times of the time cost for initiating a virtual cube with fifteen hosts.

doPost(): This method handles HTTP POST request and is used to load WebRowSet XML format data into a virtual cube. When the amount of data is huge, we also need to wait for a long while, so, here the default timeout setting should also be changed to a relative larger value. Based on our experiment result, 500 seconds is also a proper value for the timeout here. For the HTTP POST request, the content is the WebRowSet XML format data, and the result is in plain text.

3.4.2 Implementation of Database Access Servlet

Main function: Access databases.

Applied tool: JDBC, Apache Derby.

Related class files: DataServiceImpl.java in the OLAP administrator project.

Derby database network server mode

There are two runtime modes of the Derby database: embedded mode and network server mode. In our implementation the network server mode is applied. In the network server mode, the database server has a dedicated Java Virtual Machine (JVM), several applications can access the same Derby database at the same time.

As shown in Figure 3.10, a Derby network server runs as a single Java process, it

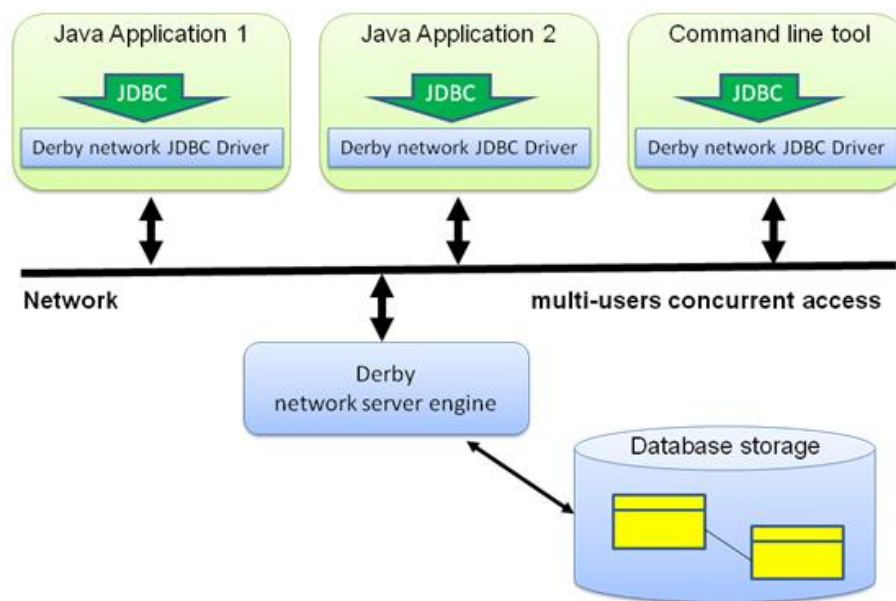


Figure 3.10: Derby network server mode

listens the client's connect through the network. The network server can accept multiple client connections and access the database storage.

role of the related file

The DataServiceImpl class file in the OLAP administrator project has two roles. On one hand, the Derby network JDBC driver is applied, and it works as client of the Derby network server. On the other hand, it implements the DataService RPC interface for the GWT client tier.(Figure 3.11)

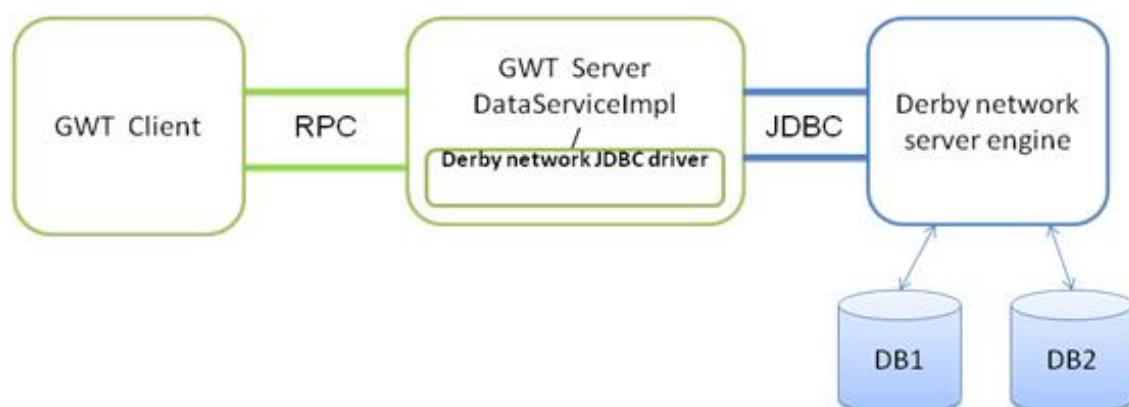


Figure 3.11: Role of the DataServiceImpl class in the project

Implementation details of the DataServiceImpl class in OLAP administrator project

Simply, the DataServiceImpl class has three responsibilities: a. Connect to databases. b. Initiate and send SQL queries. c. Handle the results.

1. Connection object

A connection object represents the connection to a specific database. the DriverManager.getConnection method is called for establishing the connection, it consumes string type parameter URL. The DriverManager class will connect to the database associated with the URL, it has a list for registered database drivers, when the method getConnection() is called, the DriverManager class will iterate the list to find the proper driver:

```
Connection conn = null;
conn = DriverManager.getConnection(url+";create=true");
```

The URL from above code is given as URL using JDBC protocol, it helps the driver to identify the desired database.

A JDBC URL (e.g. jdbc:derby//192.168.3.129:1527/forestfireDB1) has following three parts which are separated by ":".

- JDBC protocol
- Sub protocol for the specific database driver, e.g. derby
- Network data source location, e.g. //192.168.3.129:1527/forestfireDB1

2. DriverManager class

DriverManager is in the JDBC management tier between the client and database driver. It uses the available driver and establish connection to databases.

DriverManager class has a list of Driver class which were registered by calling the DriverManager.registerDriver() method. In our implementation Driver class is not directly registered by this method, instead, Class.forName() is called to explicitly register the driver by its class name to DriverManager.

```
String driver = "org.apache.derby.jdbc.ClientDriver";
Class.forName(driver).newInstance();
```

The registered drivers can be used to establish connections to databases. When the DriverManager.getConnection() method is called, DriverManager will find the first proper driver int the registered driver list and further call the Driver.connect() with the user specified URL.

3. Use WebRowSet object to conFigureSQL Statement object

After a connection is established, SQL statement can be sent to the database.

```
WebRowSet webRS = new WebRowSetImpl();
webRS.setCommand(statement);
webRS.execute(conn);
```

Here, the WebRowSet object [Ora10] is applied, it is firstly set with a SQL statement by calling the setCommand() method. Then the execute() method is called to execute the SQL statement through the already established database connection, the resulted data will be stored in the WebRowSet object.

4. Convert the WebRowSet object to WebRowSet XML

```
StringWriter sw = new StringWriter();
webRS.writeXml(sw);
response = sw.toString();
```

3.4.3 Implementation of OGSA-DAI Access Servlet

Main function: Interact with the OGSA-DAI server to get integrated data.

Applied tool: OGSA-DAI.

Related class files: ogsaDaiServiceImpl.java in the OLAP administrator project.

OGSA-DAI basic definitions

The Open Grid Services Architecture - Data Access and Integration (OGSA-DAI) provides the OGSA consistent methods to access heterogeneous data sources. It allows the client program to submit query document, so enables the remote access to the data sources. The data access and integration is achieved by providing uniform service interfaces, so multiple heterogeneous data sources could be seen as a single data source as a whole.[oE10]

Grid Data Services (GDS): This service enables data access to a resource.

Grid Data Services Factory (GDSF): The GDS service instance can be generated from the factory.

Service Group Registry (DAISGR): This is the discovery mechanism to find GDS or GDSF.

Perform Document: XML format document, used to define GDS activities, e.g. execution of an SQL statement, and used to define how to return the result.

Response Document: XML format document, represents the GDS activity execution result.

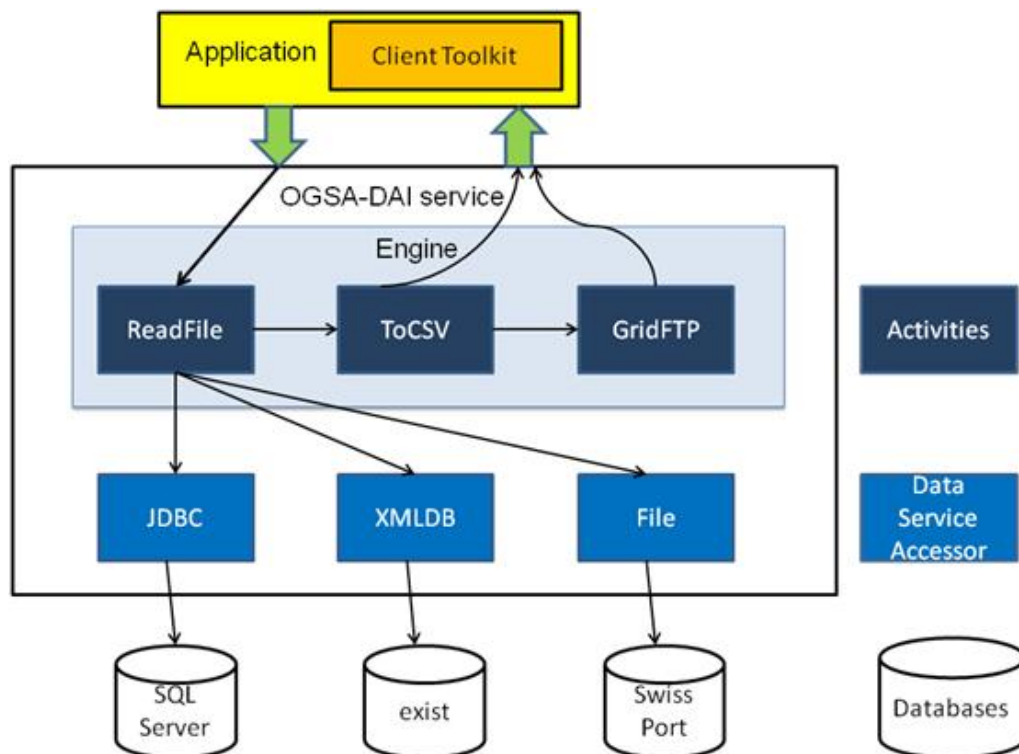


Figure 3.12: OGSA-DAI architecture

OGSA-DAI architecture

OGSA-DAI is a service-based architecture for database access over the Grid [MA05], the architecture contains the following functional parts: application, data service, activity, data service accessor and data resource. (Figure 3.12)

- Application:** This refers to the OGSA-DAI client applications, through it users can submit perform documents and get result from the server. There three kinds of actions which can be described by perform document: query (for query and update data), transform (transform data into specific format), deliver (deliver the processed data result). The query action assembles the query result in a result set and transfers it to the transform action. The transform action transforms the result set and save it as XML document which will be passed over to deliver action and finally returned to the user.
- Data service:** This is the component which directly interacts with the client application, it provides a document oriented interface, send and receive perform documents, including client's query and result, so to interact with the client.
- Activity:** When a perform document is submitted to the data service, the data service will forward the document to the related data resource. Then, the execution engine will execute the in the perform document described action

and interact with the backend data source. Finally, the data service will generate a document describing the result, and deliver it to the client.

- **Data service accessor:** For executing actions, the execution engine needs to access data resources, every data resource has its own data service accessor. E.g. JDBC for the relational databases and XMLDB for the XML database.
- **Data resource:** Heterogeneous data sources can distributed across the network, different kinds of data sources are exposed uniformly as OGSA-DAI data resources.

OGSA-DAI workflow

OGSA-DAI uses document oriented interface to support interaction with data sources. Client does not need to directly interact with the data source, but send document to the OGSA-DAI data service resource which associated with a backend data source (e.g. relational database). The document sent to the OGSA-DAI data service resource will be parsed and the described action will be performed. Then, the data service resource generates the result document and return the document through the data service back to the client. (Figure 3.13)

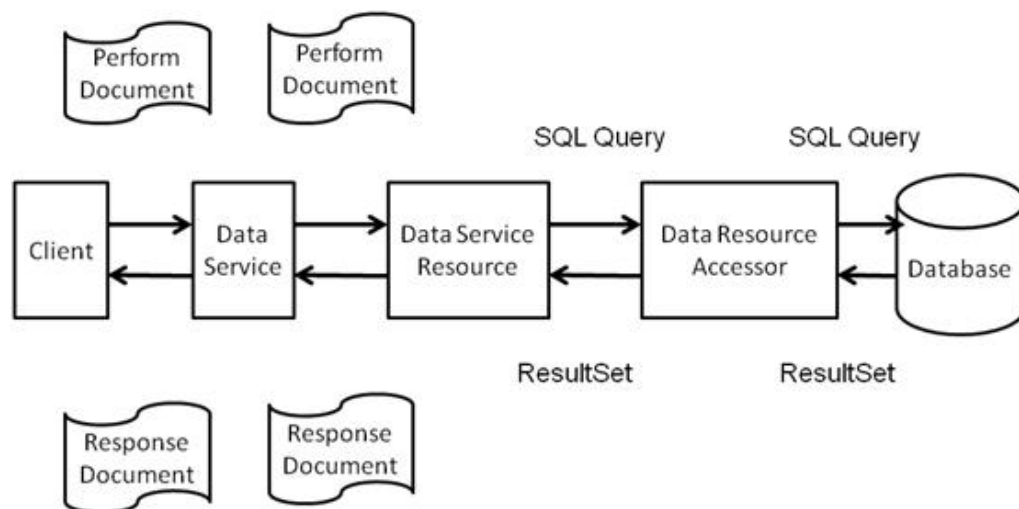


Figure 3.13: OGSA-DAI workflow

After an OGSA-DAI server is started:

1. First the GDSF starts, the DAISGR performs the registration process for it and according to the static configuration file provides MetadataExtractor class to data service. GDSF will also create objects according to data resources and then the client decide which GDSF from DAISGR to choose.
2. After the proper GDSF is chosen, a GDS instance will be generated for a certain data resource.

3. Client submits perform document to GDS, according to the perform document GDS forwards it with context parameters to the execution engine.
4. The perform document will be parsed and the described activities will be arranged in a workflow pipeline.
5. Each activity in the pipeline will be assigned to a processor and executed by the processor. Data will be transferred and processed across the pipeline.
6. Data service resource integrates the output information of the pipeline to form a result document.
7. The result document is returned to the GDS and finally forwarded to the client.

A data service has multiple data service resources, and a data service resource can only associate with one data source, data service resource interacts with its data source through the data resource accessor. As shown in Figure 3.14, a data service has a related configuration file which specifies the supported activities, session information and the class names of the data resource accessors. The configuration file of data resource accessor contains detailed description for the related data resource. The data service and the related data service resources should be located on the same server, but the residence of data resource is not restricted.

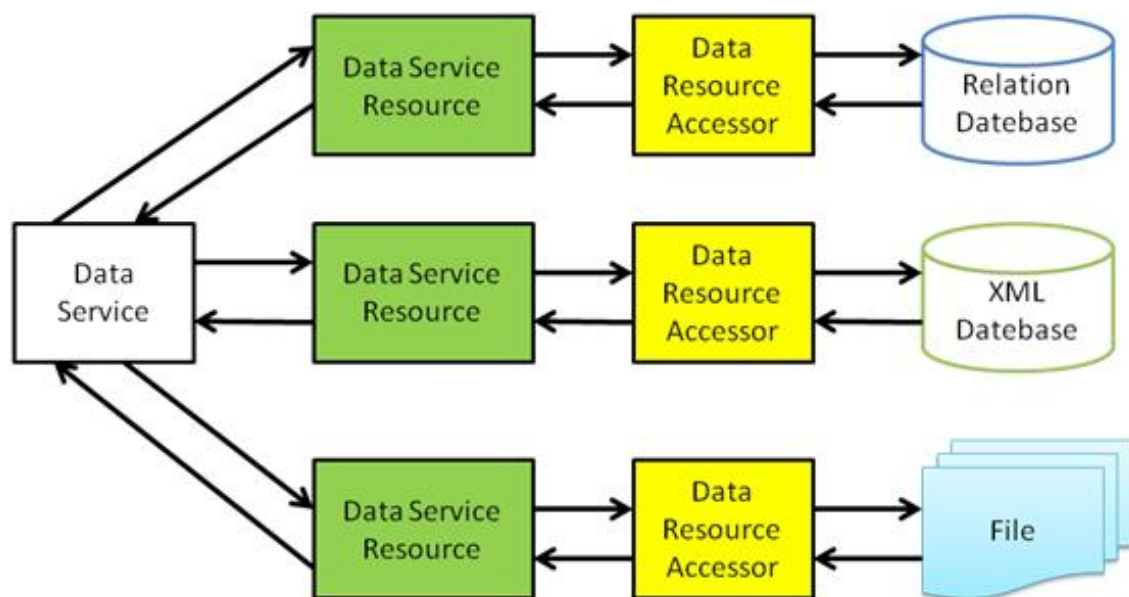


Figure 3.14: Data service, data service resources and data resources

Role of the related file

The `OgsaDaiServiceImpl` class in the OLAP administrator project has two roles to play. On one hand, it works as client of OGSA-DAI server. On the other hand, it

implements the OgsaDaiService interface for the GWT client tier RPC call. (Figure 3.15)

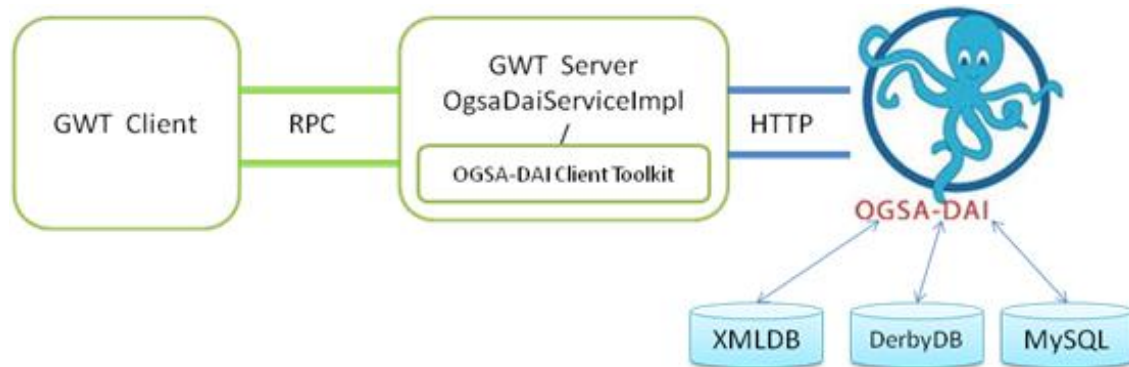


Figure 3.15: Role of OgsaDaiServiceImpl class in OLAP administrator project

Implementation details of OgsaDaiServiceImpl class

Next, let's introduce how to establish the OGSA-DAI client by applying the OGSA-DAI client toolkit which provides client side Java API for different OGSA-DAI components including services, resources and activities.

a. Tasks overview of the OgsaDaiServiceImpl class is listed below:

1. Execute the SQL statement and get the result set from database.
2. Convert the result set to WebRowSet XML format.
3. Integrate the data from the step 2.
4. Add the integrated result to request status and return to the client.

b. To fulfill the above tasks, there are some prerequisites:

1. There is an available OGSA-DAI server which could be accessed through a service URI like `http://localhost:8080/dai/services/`.
2. There are several deployed data resources on the OGSA-DAI server. E.g. the data resource FFDB1, FFDB2, DQP1 as described in Chapter 3.
3. There are some available data tables in the data resources. E.g. the table forestFireDB1 in the resource FFDB1 as described in Chapter 3.
4. OGSA-DAI client toolkit should be imported into the project.

c. Implementation steps:

1. Get a server proxy.

The code listed below:

```
ServerProxy server = new ServerProxy();
String url = "http://localhost:8080/dai/services/";
server.setDefaultBaseServicesURL(new URL(url));
```

With the above code, the client toolkit will create a serverproxy, which is used for handling the communication with the OGSA-DAI server and provides proxy to the resource on the server. Then, a Data Request Execute Resource (DRER) is needed, which receives and executes the OGSA-DAI workflow submitted by the client:

```
DataRequestExecutionResource drer=null;
drer = server.getDataRequestExecutionResource(new
ResourceID("DataRequestExecutionResource"));
```

2. Create activities in the client.

Every activity in the OGSA-DAI server side has a related activity which can be created using the client toolkit. Activity is the smallest unit of the OGSA-DAI workflow, so we can use activities to assemble workflow. As the activities are Java classes, we should instantiate them:

```
//create SQL query
SQLQuery query = new SQLQuery();
//result to WebRowset
TupleToWebRowSetCharArrays tupleToWebRowSet = new
TupleToWebRowSetCharArrays();
//resize the char array
CharArraysResize resize = new CharArraysResize();
//establish request status
DeliverToRequestStatus deliverToRequestStatus =
new DeliverToRequestStatus();
```

3. Configure and connect the activities.

When the needed activities were created, the input of them can be configured to connect either to the client's input or output of another activity. Besides, some activities focus on some certain resources, so these resources should also be specified. Next, for executing the SQLQuery activity, a related relational data resource is needed:

```
query.setResourceID(resourceId);
```

resourceId is given by the user through the OLAP administrator GUI, and transferred to the activity object as parameter. The SQLQuery activity executes the SQL statement, the method addExpression() can be used to convert plain text to SQL statement:

```
query.addExpression(statement);
```

Statement is also given by user through the OLAP administrator GUI (e.g. `select * from forestFire1`). Next, the `TupleToWebSetCharArray` activity will transform the output tuples of the `SQLQuery` activity to XML char array. Before the transformation, the output of `SQLQuery` activity should be connected to the input of `TupleToWebSetCharArrays`:

```
tupleToWebRowSet.connectDataInput(query.getDataOutput());
```

Next, the result of the `TupleToWebCharArray` activity should be integrated and result should be resized:

```
resize.connectDataInput(tupleToWebRowSet.getResultOutput());
resize.addSizeInChars(5000);
```

From above `CharArrayResize`, we can see that the data was processed in three phases: concatenation \Rightarrow resize the char array \Rightarrow execution and output. Finally, the result set will be added to request status and returned to the client:

```
deliverToRequestStatus.connectInput(resize.getResultOutput());
```

4. Create workflow.

After `conFigureand` create the needed activities respectively, next, the activities should be arranged in a pipeline workflow. How the data flows in the pipeline depends on the ordering of the activities. First, a pipeline object should be instantiated and then add the required activities to it.

```
PipelineWorkflow pipeline = new PipelineWorkflow();
pipeline.add(query);
pipeline.add(tupleToWebRowSet);
pipeline.add(resize);
pipeline.add(deliverToRequestStatus);
```

5. Execute workflow.

The in step 4 established workflow can be submitted to the DRER:

```
RequestResource requestResource=
drer.execute(pipeline, RequestExecutionType.SYNCHRONOUS);
```

We specify the execution mode of the DRER using the parameter `RequestExecutionType.SYNCHRONOUS` which indicates that the result is returned after the completion of the execution. The status of the workflow execution will be included in the request status, client can access it using a request resource proxy and can print the status:

```
RequestStatus requestStatus = requestResource.getRequestStatus();
System.out.println(requestStatus);
```

Another way is to print the request execution status description like "started", "completed" or "error" etc.

```
RequestExecutionStatus requestExecutionStatus
=requestStatus.getExecutionStatus();
System.out.println(requestExecutionStatus);
```

Visualization of the workflow

When some workflow is submitted by the client, on the OGSA-DAI server we can observe the visualized execution diagram of the workflow. As an example, let's perform a query on the DQP resource which associated with another two relational database resources, FFDB1 and FFDB2. In the OLAP administrator we can input such a query, which queries the DQP resource and get a result set as an union of the result sets from the two database resources:

```
select X,Y,MONTH,DAY,RAIN,WIND,TEMP,RH,AREA from
FFDB1_FORESTFIRE1 p join FFDB2_FORESTFIRE2 r
on p.RECORDID=r.RECORDID
```

Figure 3.16 illustrates the visualized workflow of a DQP query, it can be described as following:

1. The SQLQuery activity is executed on two database resources respectively. From the resource FFDB1 the column X, Y, MONTH and DAY are selected as result set, from the resource FFDB2 the column RAIN, WIND, TEMP, RH and AREA are selected as result set.
2. MetadataRename is executed on the two result sets, adding the resource name to each column, e.g. rename ForestFire1.DAY to FFDB1_ForestFire1.DAY.
3. According to the given condition FFDB1_Forestfire1.RECORDID=FFDB2_Forestfire2.RECORDID join the two result sets as a united result set.
4. Execute TupleArithmeticProject activity which projects columns according to a given set of arithmetic expressions, we have not specified any arithmetic expressions, so there is no changes on the result set.
5. Execute metadataRename again on the column names of the result set, e.g. FFDB1_Forestfire1.DAY to p.DAY, FFDB2_Forestfire2.AREA to r.AREA.
6. Execute SQLQuery again on processed result set from step 5 to generate the DQP result set.
7. Execute TupleToWebRowSetCharArrays activity which transform the result set into WebRowSet format.

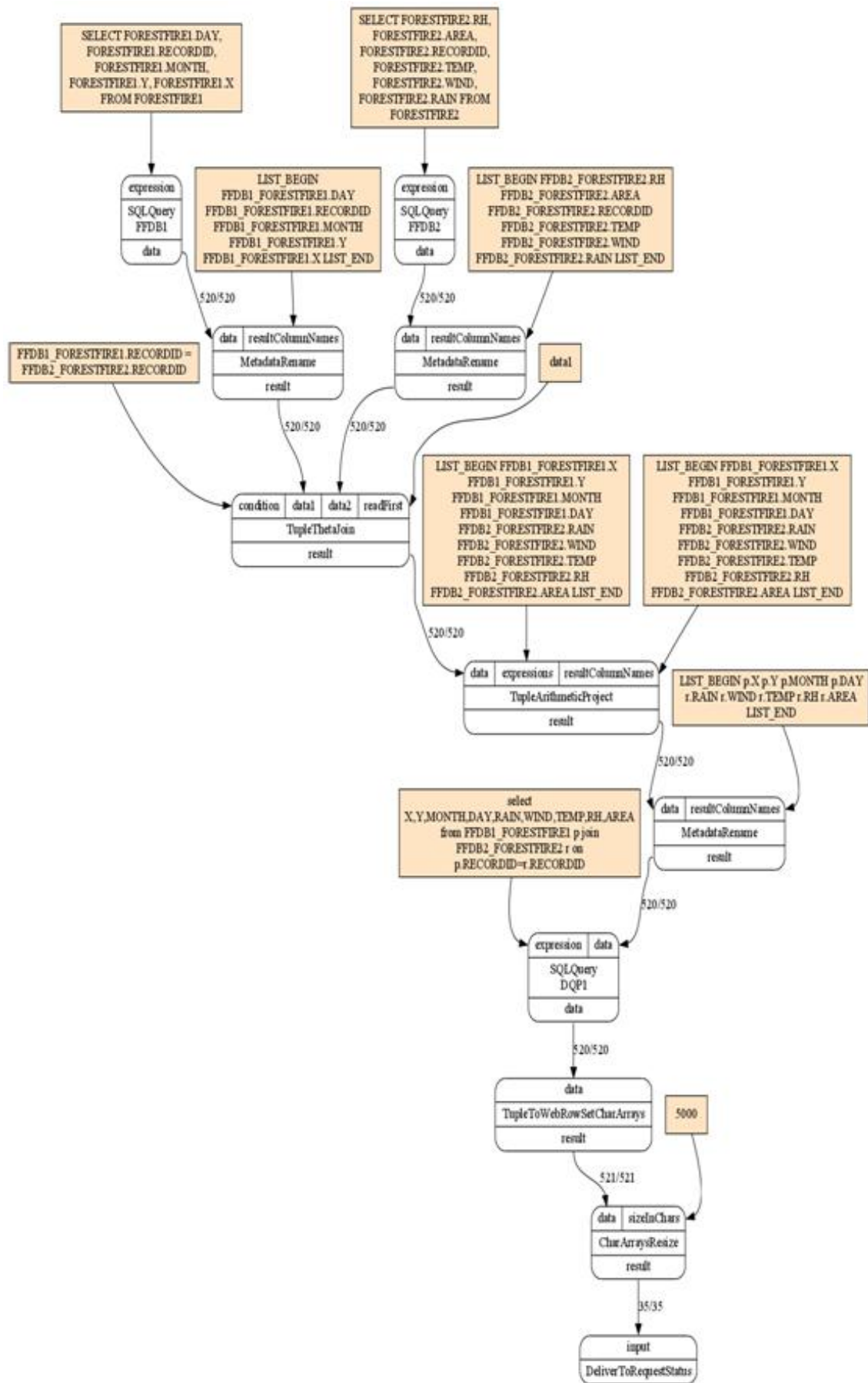


Figure 3.16: Workflow visualization of a DQP query

8. Resize the char array with CharArraysResize activity.
9. Deliver the result set to request status by executing DeliverToRequestStatus activity.

3.4.4 Implementation of the WebRowSet XML Generator

Main function: Transform the result set into WebRowSet XML format.

Applied tool: WebRowSet implementation from sun

Related file: The WebRowSet XML generator is embedded in OgsaDaiServiceImpl.java and DataServiceImpl.java in OLAP administrator project.

In this section we will first introduce the inheritance structure of the WebRowSet interface, and discuss why it is applied here. then, the implementation of the functional module, WebRowSet XML generator, is described.

Inheritance structure of the WebRowSet interface

WebRowSet is an interface from the JDBC API, it helps developer efficiently interact with the result data set from the database query, efficiently output data set to or read data from a corresponding WebRowSet XML document.

The inheritance structure diagram is given in Figure 3.17. The root is the java.sql.ResultSet interface, whose implementation stands for the result set of table formatted data. This kind of data can be generated by executing java.sql.Statement. The result set can only be forward iterated and can not be updated, so it also implies low performance for precise data access control.

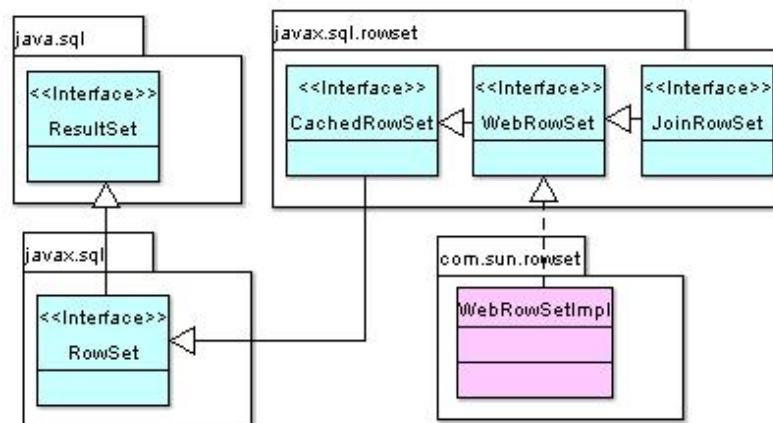


Figure 3.17: Inheritance structure of JDBC WebRowSet interface

Thus, which choice do we have depends on which operations should be performed on the data set. For example, if the data set should support the JavaBeans component model JDBC API, the sub interface javax.sql.RowSet should be applied. As access database from a Java program is a heavyweight operation, the data cache in the

X	Y	MONTH	DAY	RAIN
4	4	sep	thu	0.0
4	3	aug	sun	0.0
4	3	aug	wed	0.0
4	3	aug	wed	0.0
4	3	aug	thu	0.0

Table 3.2: Database query result

main memory is a key factor, so if the data set should be saved in the data cache container in memory, we can apply the sub interface `javax.sql.rowset.CachedRowSet` whose implementation enables the ability to operate on data from the data source without having to constantly connect to it. Moreover, `CachedRowSet` could be better iterated and could be serialized. Besides the features provided by `CachedRowSet`, in our application we also need to output the result data set to XML document, as a result, `javax.sql.rowset.WebRowSet` becomes our best choice. The sun's implementation `com.sun.rowset.WebRowSetImpl` is applied for the `WebRowSet` interface.

Details of the WebRowSet XML generator

The `WebRowSet` XML generator module is embedded both in the DB access Servlet and OGSA-DAI access servlet to facilitate the transformation of result set into `WebRowSet` XML. Next, let's introduce a simple example. With the query statement: `select X, Y, MONTH, DAY, RAIN from forestFireDB1` we can get the result set from the database as shown in Table 3.2.

First, we need to use a `ResultSet` object to get the SQL statement execute result. Then the `ResultSet` object is converted to a `WebRowSet` object, which can help us output the result to XML document efficiently:

```
try {
    ...
    ResultSet resultSet = stmt
        .executeQuery("select X,Y,MONTH,DAY,RAIN from forestFireDB1");
    WebRowSet wrs = new WebRowSetImpl();
    wrs.populate(resultSet);
    StringWriter sw = new StringWriter();
    wrs.writeXml(sw);
    s = sw.toString();
    resultSet.close();
} catch (DataStreamErrorException e) {
    e.printStackTrace();
    ...
}
```

The `wrs.writeXML()` method from above code will generate a `WebRowSet` XML document in compliance with the `WebRowSet` XML Schema Definition (XSD). The

output document includes three parts: properties, metadata and data. The structure is given below:

```
<?xml version="1.0"?>
<webRowSet xmlns= "http://java.sun.com/xml/ns/jdbc"
xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation= "http://java.sun.com/xml/ns/jdbc
http://java.sun.com/xml/ns/jdbc/webrowset.xsd">
  < properties>
    ...
  </properties>
  <metadata>
    ...
  </metadata>
  <data>
    ...
  </data>
</webRowSet>
```

Within the <properties> tag there is the property description about the data set. Within the <metadata> tag there is the metadata of the data set, including record counts, column type, column name etc.

Within the <data> tag there is the data "payload" - Rows of records are listed one after another as following:

```
<data>
  <currentRow>
    <columnValue>4</columnValue>
    <columnValue>4</columnValue>
    <columnValue>sep</columnValue>
    <columnValue>thu</columnValue>
    <columnValue>0.0</columnValue>
  </currentRow>
  <currentRow>
    <columnValue>4</columnValue>
    <columnValue>3</columnValue>
    <columnValue>aug</columnValue>
    <columnValue>sun</columnValue>
    <columnValue>0.0</columnValue>
  </currentRow>
  ...
</data>
```

Within a <currentRow> tag there is one row of data which is analog to one row of records in the database.

3.4.5 Convert the WebRowSet XML Document to Ext GWT Grid

Main function: Present the WebRowSet XML document well in a Ext GWT grid.

Applied tool: Ext GWT.

Related method: `parserXML()` and `createGrid()` method of the Administrator class in the OLAP administrator project.

Related class file: `Record1.java`, `Record2.java`, `Record3.java`, `Record4.java`, `Record5.java`, `Record6.java`, `Record7.java`, `Record8.java`, `Record9.java` and `Record10.java` in the OLAP administrator project.

There is a WebRowSet XML parser module in the GWT client tier, it is used to parse the WebRowSet XML document from the GWT server tier and generate related record object to facilitate the instantiation of Ext GWT grid object [Sen]. Here, Ext GWT grid refers to a GUI component from the Ext GWT framework.

Besides well presenting a data set in a grid table, the Ext GWT grid also enables some useful operations on the data set including sorting, editing, filtering and so on. So, these operations could be performed as preprocess on the data set by the user through the OLAP administrator GUI before loading the data set to the OLAP cloud.

The following part of the section will introduce the processes of how to convert a WebRowSet XML document to a Ext GWT grid.

ColumnModel `parserXML(String messageXml)`: Before we use the Ext GWT grid, column information should be defined, this method is used to get the column-name information from the WebRowSet XML document:

```
List<ColumnConfig> columns = new ArrayList<ColumnConfig>();
    for(int i=0;i<nl.getLength();i++){
        // create configuration for each columne
        columns.add(new ColumnConfig("id"+(i+1),
nl.item(i).getFirstChild().getNodeValue(), 100));
    }
    // create model for all columes in the table
    ColumnModel cm = new ColumnModel(columns);
```

The column definition is stored in ColumnModel. The constructor of the ColumnModel takes `list<ColumnConfig>` as parameter which defines column information (id, title, width etc.) of a column. id is a property of BaseModel class.

`createGrid(ColumnModel cm ,String messageXml)`: After `conFigurecolumns`, this method is called to create grid object. `ListStore` and `ColumnModel` from above is used to generate the needed grid:

```
Grid<BaseModel> grid = new Grid<BaseModel>(records,cm);
```


There are two parameters from above, `cm` is object of `ColumnModel` which was generated by `parserXML()` method. The parameter `records` represents an object of class `ListStore<BaseModel>`. `ListStore<BaseModel>` is an important data storing type in Ext GWT, apparently, we need to generate the corresponding `ListStore<BaseModel>` object in order to convert the `WebRowSet` XML document to Ext GWT grid.

Usage of related classes: `Record1.java`, `Record2.java`, ..., `Record10.java`

Based on the description of the above two methods, we can see that `WebRowSet` XML parser module's key function is to implement data storing object `ListStore<BaseModel>` for storing the data set. First, we should have to introduce how Ext GWT handles data.

The essential part of the Ext GWT data handling mechanism is the abstract class `Store`. The `Store` class has two sub classes, `ListStore` and `TreeStore`. `ListStore` for storing column-wise data, `TreeStore` used for storing tree structure data.

Abstract class `ListStore` is generic class, its generic parameter is the `ModelData` interface or the sub classes. The `BaseModel` class is an implementation of the Ext GWT `ModelData` interface. E.g. in our application, class `Record1`, `Record2`, ..., `Record10` are sub classes of class `BaseModel`. A code example of class `Record2` is shown in the right part of Figure 3.18.

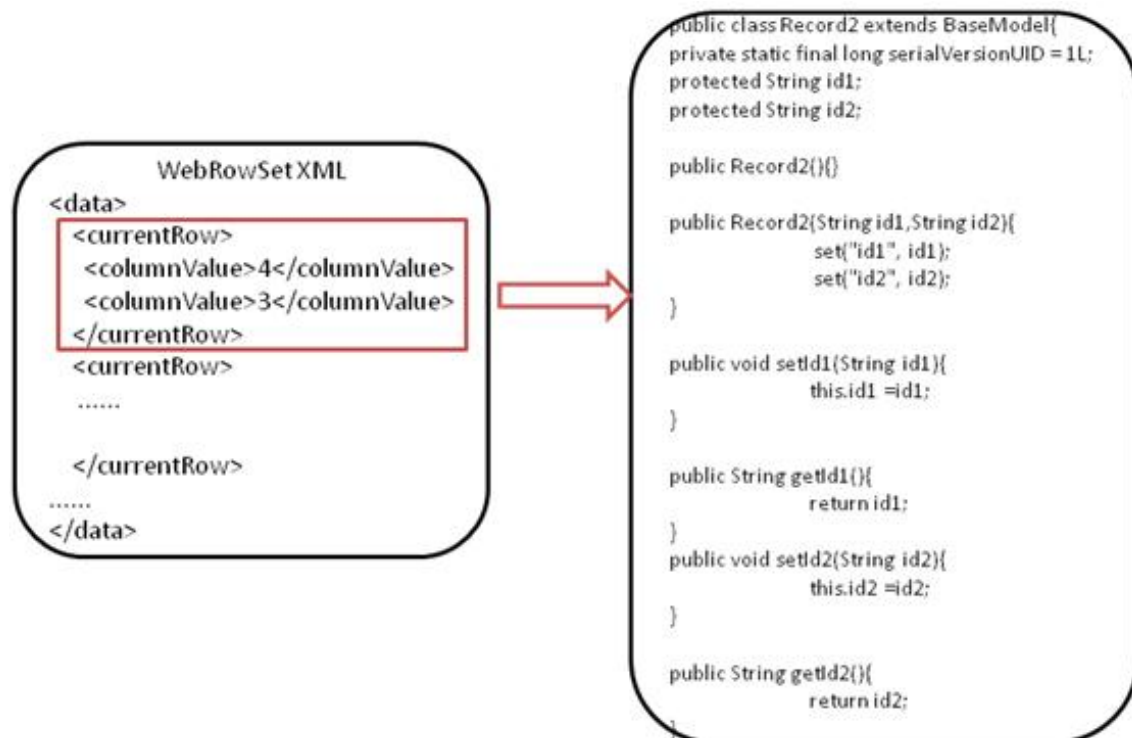


Figure 3.18: Convert `currentRow` to an object of `Record2` class

Because in `WebRowSet` XML, `<currentRow>` represents one row of record in data

table, in this data converting task, the main task is to implement the conversion for each `<currentRow>` to a proper data storing object by using the `Record1`, `Record2`, ..., `Record10` classes as shown in Figure 3.18.

Next, all the generated data storing object (e.g. all the `Record2` objects) should be added to an object of class `ListStore<BaseModel>` which represents all `<currentRow>` in the `WebRowSet XML`. By storing the data in an object of `ListStore<BaseModel>` and applying the above mentioned two methods, we can implement the conversion from `WebRowSet XML` to `Ext GWT grid`.

3.4.6 Modules in GWT Development

Related file: `Administrator.gwt.xml` in OLAP administrator project and `OLAP-QueryClient.gwt.xml` in OLAP query client project.

Introduction of modules

A configurable GWT component is called GWT module. In GWT, configuration information for compiling a project is stored in a module definition file, which includes module entry point, module inheritance information, path of source code, resource file path and timeout binding rules.

File name extension of a GWT module definition file is `.gwt.xml`. Module name is made up by its package name plus the definition file name. E.g. in the package `com.google.gwt.admin` of OLAP administrator project, there is a `Administrator.gwt.xml` file which is the module definition file of the project, the module name is `com.google.gwt.admin.Administrator`. For compiling a GWT project, the compiler should have to know which GWT modules should be compiled. If we open the run configuration window in Eclipse and select the GWT tab we can see `Administrator-com.google.gwt.admin` is listed in the Available Modules list. (Figure 3.19)

Following is the content of `Administrator.gwt.xml` file from the OLAP administrator project:

```
<module>
  <inherits name="com.google.gwt.user.User"/>
  <inherits name="com.google.gwt.user.theme.standard.Standard"/>
  <inherits name="com.google.gwt.xml.XML" />
  <inherits name="com.extjs.gxt.ui.GXT"/>
  <stylesheet src="../../gxt/css/gxt-all.css"/>
  <entry-point class="com.google.gwt.admin.client.Administrator"/>
</module>
```

The definition file is an XML document with the root element "module". After compilation of the module, the generated files are stored in the folder (folder name is same as module name) under the *war* folder of the project, in this example the generated files are stored in `com.google.gwt.admin.Administrator` folder.

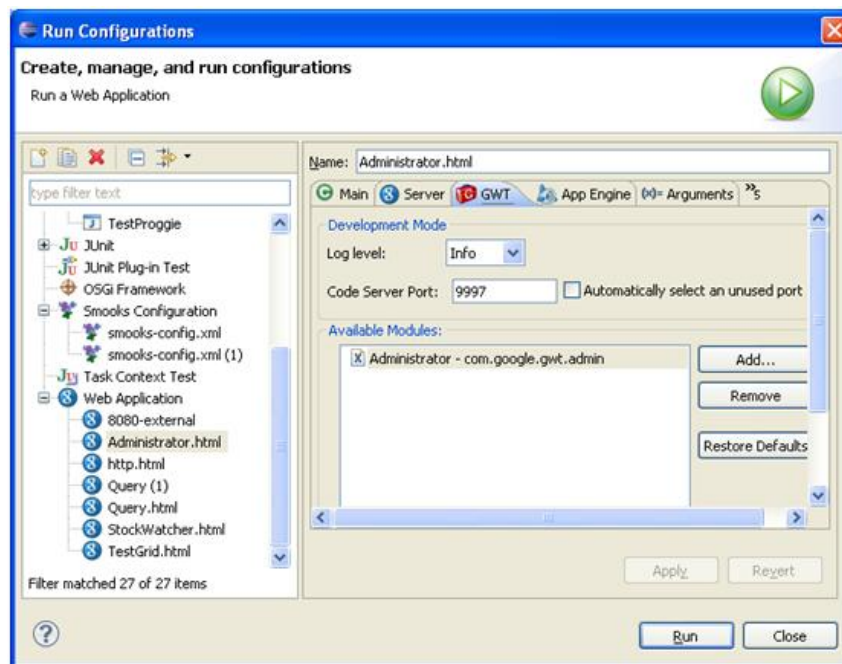


Figure 3.19: GWT run configuration window

Module entry point

When use JavaScript to write AJAX application, window onload event is the entry point of the application. in GWT module there is also an entry point. The class which implements the EntryPoint interface can be the entry point of an GWT application:

```
package com.google.gwt.admin.client;
public Interface EntryPoint {
    void onModuleLoad();
}
```

In the EntryPoint interface the only method to be implemented is onModuleLoad(). This method is called right after the module is loaded. The class which implements the EntryPoint interface should also be registered in the GWT module definition file, whose entry-point element refers to the entry point, the class attribute indicates the class name of the entry point class. In the method onModuleLoad() of the entry point class, there are code for all the graphical components and events to be listened. This code is arranged in the onModuleLoad() method but not in the constructor of the entry point class, because it not possible to guarantee that when the constructor is called, the whole page and all its components are already loaded.

Module inheritance

Module configuration could be simplified by using inheritance. In module definition file use inherits element to indicate inheritance relationship to other modules, the name attribute refers to the module name of the other module. The Administrator module inherits four other modules:

```
<inherits name="com.google.gwt.user.User"/>
<inherits name="com.google.gwt.user.theme.standard.Standard"/>
<inherits name="com.google.gwt.xml.XML" />
<inherits name="com.extjs.gxt.ui.GXT"/>
```

If a module from the GWT library is to be applied, it should be inherited and defined in the module definition file. If there is a error "No source code is available ..." appears in the GWT runtime console, then it is possible that the inheritance to the module is not defined.

Normally, com.google.gwt.user.User should be inherited, as it contains all GWT essential components. The appearance mode of GWT application is also implemented as module, there are three available appearance modes: Chrome, Dark and standard. The example from above uses standard appearance mode. The third module which should be inherited is com.google.gwt.xml.XML which is responsible for handling XML in GWT application. For example, in my implementation this module is applied for implementing the WebRowSet XML and OMML parsers. The fourth module which should be inherited is com.extjs.gxt.ui.GXT, this module is needed for Ext GWT plugin.

Adding extra CSS

Sometimes, a GWT module need to refer external CSS file. If a reference CSS locates in a HTML file, then it breaks the good practise, tight internal coupling and loose external coupling. Besides, if the GWT module is referred by several other applications, each of these applications should add reference to the CSS file in every page which uses the module, and this apparently undermines the principle, don't repeat yourself. The solution of GWT is to define the reference to the CSS in the module definition file. During initiation of a module, the CSS will be loaded to related pages. For example:

```
<stylesheet src="../gxt/css/gxt-all.css"/>
```

The stylesheet element guides the GWT compiler to add the CSS file (indicated by attribute src) to related pages. The CSS from above (gxt-all.css) is needed by component in Ext GWT style.

3.4.7 Implementation of PRC Call

There three possibilities for the GWT client tier to communicate with the GWT server tier: XMLHttpRequest, JSON, GWT-RPC. In our implementation GWT-RPC is applied. There are following advantages to apply GWT-RPC:

First, because the RequestBuilder.sendRequest() method can only send string data, no matter XMLHttpRequest or JSON, serialization and deserialization for the data are always needed both at the client and the server tier. The GWT-RPC encapsulates the serialization and deserialization work which is not to be concerned

by developer. Besides metadata will also be serialized and deserialized by XML-HttpRequest and JSON method, e.g. the metadata add, a and b in the following example:

```
<add>
  <a>3</a>
  <b>4</b>
</add>
```

The GWT-RPC only serialized and deserialized data but not metadata, so the generated string should be small and so low communication cost. Next, let's describe the RPC call in our implementation.

RPC remote interface

Related interfaces: OLAPService.java in the OLAP query client project, OLAPService.java, OgsaDaiService.java and DataService.java in the OLAP administrator project.

First RPC interface should be defined, so that the server program can implement the required functions according to the interface definition. For example, in the OLAP query client project, there is a RPC interface OLAPService in the package com.google.gwt.query.client. The interface is used to send different HTTP requests to the OLAP cloud and handle responses:

```
@RemoteServiceRelativePath("OLAP")
public interface OLAPService extends RemoteService{
    String doGet(String input);
    String doQuery(String url,String input);
    String doPut(String url,String input);
}
```

The interface should inherit RemoteService interface and also annotated with @RemoteServiceRelativePath, which indicates the URL path to the servlet implementation of the interface. The detailed description of the methods of this interface can be found in section 4.3.1.

Implementation of RPC interfaces

Related classes: OLAPServiceImpl.java in OLAP query client project, OLAPServiceImpl.java, DataServiceImpl.java and OgsaDaiServiceImpl.java in OLAP administrator project.

The implementation of GWT-RPC is based on servlet. In the GWT server tier there should be RPC implementation class which inherits the com.google.gwt.user.server.rpc.RemoteServiceServlet class. RemoteServiceServlet is provided by GWT and inherits the HttpServlet class. Once a client request the servlet, it will deserialize the client's request and find the corresponding method, after execution, the result will

be serialized and returned to the client. In the package `com.google.gwt.query.server`, the `OLAPServiceImpl` class implements the `OLAPService` interface, the `DataServiceImpl` class implements the `DataService` interface, `OgsaDaiServiceImpl` class implements `OgsaDaiService` interface. Below is the example of `OLAPServiceImpl` class:

```
package com.google.gwt.admin.server;
public class OLAPServiceImpl extends RemoteServiceServlet implements
OLAPService{
    ...
}
```

The servlet should be registered so to provide GWT-RPC service for client. In OLAP query client project, `OLAPServiceImpl` should be registered in `war/WEB-INF/web.xml`:

```
<servlet>
    <servlet-name>OLAPService</servlet-name>
    <servlet-class>
        com.google.gwt.query.server.
    </servlet-class>
</servlet> OLAPServiceImpl

<servlet-mapping>
    <servlet-name>OLAPService</servlet-name>
    <url-pattern>
        /com.google.gwt.query.OLAPQueryClient/OLAP
    </url-pattern>
</servlet-mapping>
```

Here, the `<servlet>` and `<servlet-mapping>` element is required. Within `<servlet>` element, `servlet-class` indicates the implementation class is `OLAPServiceImpl`. Within `<servlet-mapping>` element, `url-pattern` indicates where the interface is. this servlet should be registered at `/com.google.gwt.query.OLAPQueryClient/OLAP`. `com.google.gwt.query.OLAPQueryClient` is the output folder of the module. The name `OLAP` is annotated in the `OLAPService` interface.

Asynchronous interface

Related asynchronous interfaces: `OLAPServiceAsync.java` in OLAP query client project, `OLAPServiceAsync.java`, `DataServiceAsync.java` and `OgsaDaiServiceAsync.java` in OLAP administrator project.

The first letter A of AJAX means asynchronous. For example, we need to call the `OLAPService` interface asynchronously, thus, we have to define a related asynchronous interface, which should be in the same package as the synchronous interface and the name should be `[interfaceName]Async`. I.e. synchronous interface `OLAPService` should have a related asynchronous interface with the name `OLAPServiceAsync`, synchronous interface `DataService` should have a related asynchronous interface with the name `DataServiceAsync`:

```

package com.google.gwt.query.client;
import com.google.gwt.user.client.rpc.AsyncCallback;
public interface OLAPServiceAsync{
    void doGet(String input, AsyncCallback<String> callback);
    ...
}

```

In GWT-RPC, each synchronous interface should have a related asynchronous interface. The return type of the asynchronous interface is NULL, additionally with an AsyncCallback parameter. AsyncCallback is generic class, the generic parameter is the return value of the RPC method. So, String doGet(String input) has related asynchronous version Void doGet(String input, AsyncCallback<String> callback). The generic parameter should be a reference but not a basic data type.

Make GWT-RPC call at client side

After the above work, we can make GWT-RPC call in our GWT client tier. For example, the entry point class OLAPQueryClient in the OLAP query client project uses the doGet() method of the OLAPService interface:

```

OLAPServiceAsync olapSrv = GWT.create(OLAPService.class);
public void get(String url){
    ...
    AsyncCallback<String> callback = new AsyncCallback<String>() {
        public void onFailure(Throwable caught) {
            ...
        }
        public void onSuccess(String result) {
            ...
        }
    };
    olapSrv.doGet(url, callback);
}

```

First, GWT.create() is called to generate an instance of the Asynchronous interface. Then, we can call the OLAPServiceAsync.doGet() method, in the asynchronous version the doGet() method the second parameter is callback object. If the call succeeds, the onSuccess() method will be invoked, if the call fails, the onFailure() method will be invoked. There is a parameter for the onSuccess() method, this parameter is a generic parameter of type AsyncCallback. Indeed, it is the return value of the remote call, the return value of doGet() method.

3.5 UML Diagrams Description

In order to better understand the structure of our implementation, in this section we will give some UML diagrams and related descriptions.

3.5.1 UML Diagram of the OLAP Query Client Project

Figure 3.20 is the class diagram for the OLAP query client project. There are two packages: `com.google.gwt.query.client` package contains all the GWT client tier classes and interfaces including `CubeInformation`, `OLAPQueryClient`, `OLAPService` and `OLAPServiceAsync`. `com.google.gwt.query.server` package contains class `OLAPServiceImpl` for the GWT server tier.

`OLAPQueryClient` is the entry point class of the project, its `onModuleLoad()` is the entry point method. Class `OLAPQueryClient` uses two interfaces for making RPC call, the `OLAPServiceAsync` is the related asynchronous interface for interface `OLAPService`. In the GWT server tier there is only one class `OLAPServiceImpl`, it implements `OLAPService` interface and enables access to the OLAP cloud and return the result to `OLAPQueryClient` in the GWT client tier. After receive the result, the `OLAPQueryClient` will parse the OMMML message from the server and store the information in an object of `CubeInformation` class, which contains description for a virtual cube. Following are the detailed descriptions of the classes, their properties and methods:

- **Class `CubeInformation`:** contains description for virtual cube.
 - Properties:**
 - `cubeID`:** The unique ID of a virtual cube in the OLAP cloud
 - `dimName`:** String list, each String of the list represents a dimension name of the virtual cube.
 - `dimNum`:** Number of dimension of the virtual cube.
 - `dimValue`:** List of String lists, represents all dimension members for every dimension of the virtual cube.
 - Methods:**
 - `getCubeID()`:** Get ID of the virtual cube.
 - `setCubeID()`:** Set ID of the virtual cube.
 - `getDimName()`:** Get the dimension names
 - `setDimName()`:** Set the dimension names
 - `getDimNum()`:** Get the number of dimensions
 - `setDimNum()`:** Set the number of dimensions
 - `getDimValue()`:** Get metadata of a virtual cube.
 - `setDimValue()`:** Set metadata of a virtual cube.
- **Class `OLAPQueryClient`:** entry point class, uses RPC interface `OLAPService` and `OLAPServiceAsync`.
 - Part of its properties:**
 - `url`:** Service URI of the virtual cube server in the OLAP cloud.
 - `urlVCube`:** URI of the selected virtual cube.
 - `cubesInfo`:** List of `CubeInformation` objects containing description of all available virtual cubes on the current server.
 - `queryValue`:** List of query, each element of the list is a `ListBox` object which contains all dimension members of a dimension. The selected item of the `ListBox` should be parameter of the query.
 - `arrayTextBox`:** Used to input parameter for load one row operation.

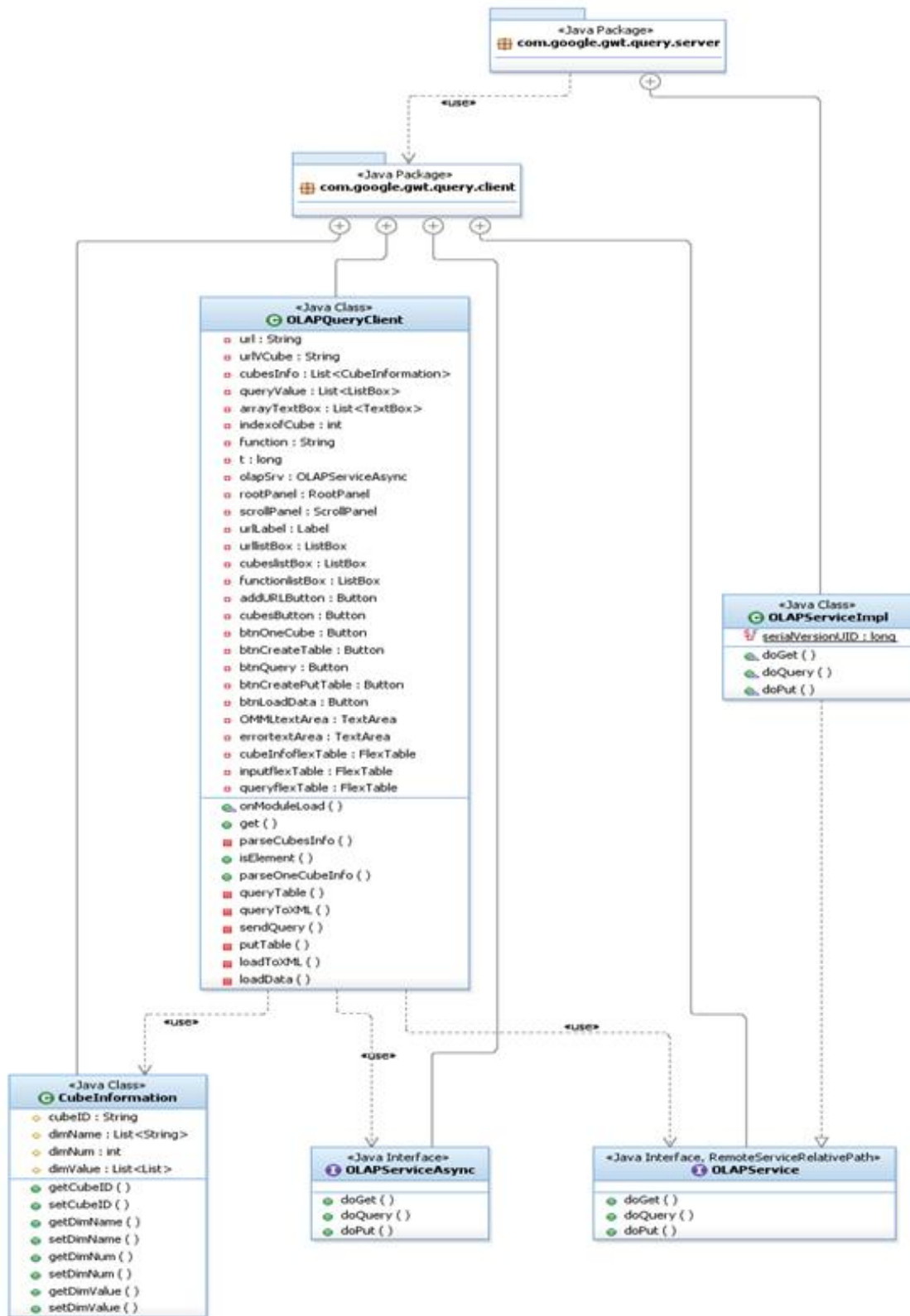


Figure 3.20: Class diagram of the OLAP query client project

indexOfCube: Index number of the currently selected cube.

function: Indicates which method should be applied for aggregation query.

olapSrv: Instance of the OLAPServiceAsync interface.

Methods:

onModuleLoad(): Entry point method of the class and of the project. It will be called at application initiation.

get(): Used to invoke the doGet() method of the OLAPServiceAsync interface.

parseCubesInfo(): Used to parse the <ServerInfo> element from OMML and put the cubeID of all the available virtual cubes to client's cubeID list.

isElement(): Used to find if the new CubeInformation object is already in the cubesInfo list.

parseOneCubeInfo(): Used to parse the <DataDictionary> element from OMML so to get the metadata of a virtual cube, store the metadata in cubesInfo list and create form in the GUI based on the metadata.

queryTable(): Get the selected cube's metadata from cubesInfo list, and create query table in the GUI which is used for initiating query to the virtual cube.

queryToXML(): Generate <Query> OMML message based on parameters in query table.

sendQuery(): Used to invoke doQuery() method of the OLAPServiceAsync interface, parse and present the returned <Result> OMML message which contains the result of the OLAP query.

putTable(): Create load one row table in the GUI.

loadToXML(): Input validation for load one row table, the input for dimension should not be empty and for measure should be double type value. The method also converts the parameters to <LoadOneRow> XML message, write it on the "XML state" panel in the GUI and invoke the loadData() method.

loadData(): Used to invoke doPut() method of the OLAPServiceAsync interface. The returned message will be write on "state" panel in the GUI.

- **Interface OLAPService:** RPC interface for the OLAPServiceImpl class.
 - doGet()
 - doQuery()
 - doPut()
- **OLAPServiceAsync interface:** Asynchronous interface for the OLAPService interface.
 - doGet(): Asynchronous method for the synchronous doGet() method of OLAPService interface.
 - deQuery(): Asynchronous method for the synchronous doQuery() method of OLAPService interface.
 - doPut(): Asynchronous method for the synchronous doPut() method of OLAPService interface.
- **OLAPServiceImpl class:** Implements the OLAPService interface and enables access to the OLAP cloud.

doGet(): Get server information and virtual cube metadata from the OLAP cloud.

doQuery(): Send OLAP query to virtual cubes, and handle the result.

doPut(): Load one row of data to a virtual cube.

3.5.2 UML Diagram of the OLAP Administrator Project

Figure 3.21 gives the class diagram of the OLAP administrator project. The class diagram just shows the structure overview without properties or methods, as the project is relative complicated. Detailed properties and methods could be found in Appendix.

There are two packages: Package `com.google.gwt.admin.server` contains class `DataServiceImpl`, class `OgsaDaiServiceImpl` and class `OLAPServiceImpl`. Package `com.google.gwt.admin.client` contains all the classes and interfaces for GWT client tier.

- **Class Administrator:** It is the entry point class of the project, it uses six RPC interfaces to communicate with GWT server tier.

Part of the Properties:

url: Service URI of virtual cube server.

urlVCube: URI of virtual cube.

data: WebRowSet XML format data.

cubesInfo: List of CubeInformation objects containing description of all available virtual cubes on the current server.

arrayTextBox: Used to input parameter for initiating new virtual cube.

olapSrv: Instance of the OLAPServiceAsync interface.

Methods:

onModuleLoad(): Entry point method of the class and of the project. It will be called at application initiation.

parseXML(): Get column-name information from metadata of WebRowSet XML to be used as column definition of Ext GWT grid.

createGrid(): It uses ListStore object and the column definition from `parseXML()` to create Ext GWT grid.

parseCubesInfo(): Used to parse the `<ServerInfo>` element from OMML and put the cubeID of all the available virtual cubes to client's cubeID list.

parseOneCubeInfo(): Used to parse the `<DataDictionary>` element from OMML so to get the metadata of a virtual cube, store the metadata in cubesInfo list and create form in the GUI based on the metadata.

isElement(): Used to find if the new CubeInformation object is already in the cubesInfo list.

inputCheck(): Used to make sure the input of create new cube table is not empty.

createMetadata(): Convert the parameters from the create new cube table to `<DataDictionary>` message, write this to "XML state" panel and invoke the `postMitHost()` method.

createTable(): Generate create new cube table in the GUI, number of rows is given by user input.

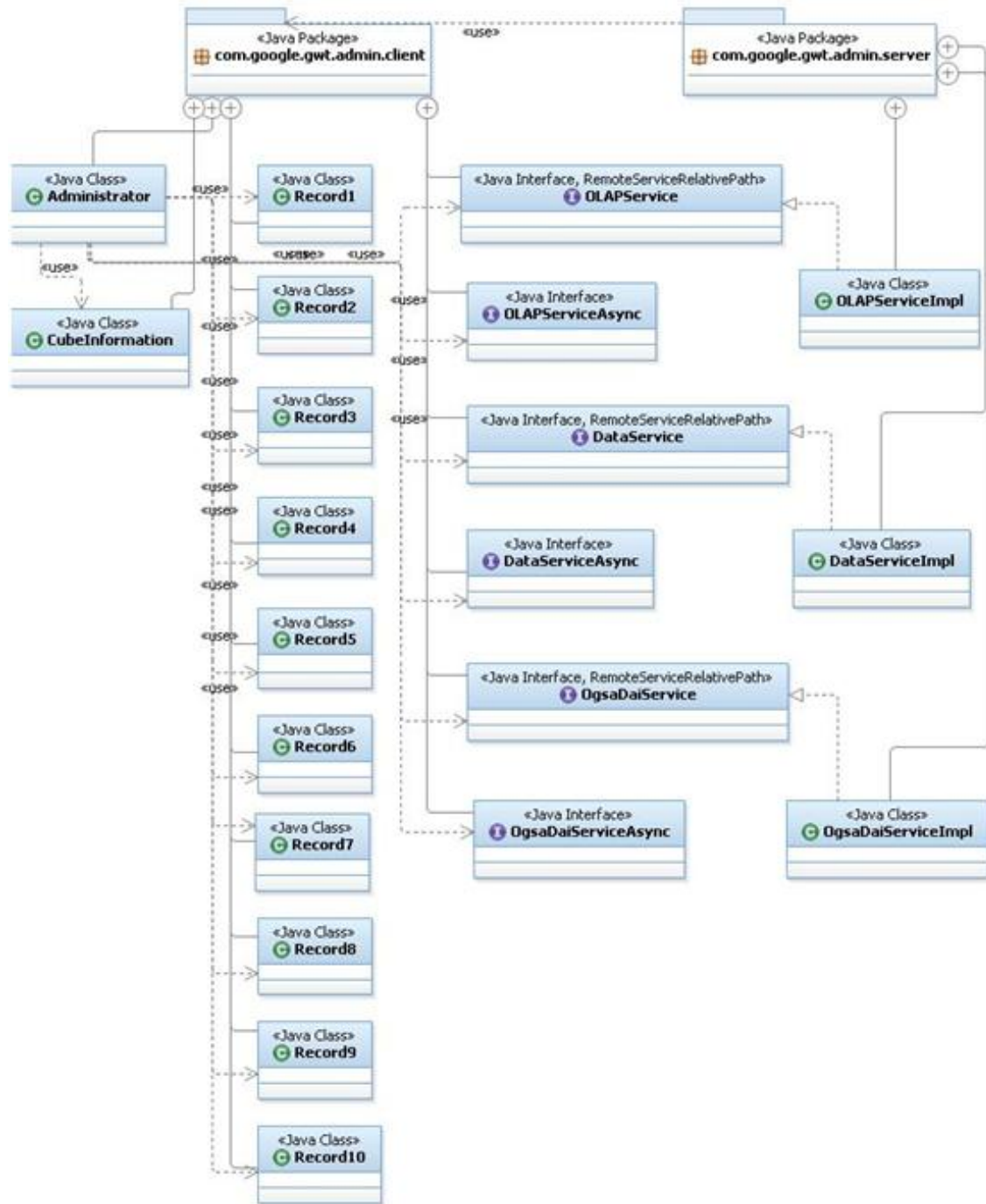


Figure 3.21: Class diagram of the OLAP administrator project

- get():** Invoke doGet() method of the OLAPServiceAsync interface. furthers in the onSuccess callback method, invoke parseCubesInfo() when query information of a server, invoke parseOneCubeInfo() when query a virtual cube.
- postMitHost():** Invoke doPostMitHost() method of the OLAPServiceAsync interface. The result will be write on "state" panel.
- post():** invoke doPost() method of the OLAPServiceAsync interface. The result will be write on "state" panel.
- delete():** invoke doDelete() method of the OLAPServiceAsync interface. The result will be write on state panel and the related CubeID will be removed from the CubeID list.
- derbyCon():** Invoke getData() method of the DataServiceAsync interface. The result, WebRowSet XML data, will be write on "XML state" panel and the createGrid() method will be invoked to present the data in Ext GWT grid.
- ogsaCon():** Invoke getData() method of the OgsaDaiServiceAsync interface. The result, WebRowSet XML data, will be write on "XML state" panel and the createGrid() method will be invoked to present the data in Ext GWT grid.
- **Class CubeInformation:** Used for storing virtual cube information like the same class in OLAP query client project.
 - **Class Record1, Record2, Record3, Record4, Record5, Record6, Record7, Record8, Record9, Record10:** These classes are similar, each of them represents one record. Let's take Record3 as example, it represents a record with three attributes, two attributes for two dimension and the third attribute for measure.

Properties:

id1: Value for the first dimension.

id2: Value for the second dimension.

id3: Value for measure. **Methods:**

Record3(): Constructor without parameter.

Record3(String id1, String id2, String id3): Constructor with parameter.

setId1(): Set value for the first dimension.

getId1(): Get value from the first dimension.

setId2(): Set value for the second dimension.

getId2(): Get value from the second dimension.

setId3(): Set value for the measure.

getId3(): Get measure value.
 - **Interface OLAPService:** RPC interface for the OLAPServiceImpl class.

doGet()

doDelete()

doPostMitHost()

doPost()

- **Interface OLAPServiceAsync:** asynchronous interface for interface OLAPService.
doGet(): Asynchronous method for the synchronous doGet() method of OLAPService interface.
doDelete(): Asynchronous method for the synchronous doDelete() method of OLAPService interface.
doPostMitHost(): Asynchronous method for the synchronous doPostMitHost() method of OLAPService interface.
doPost(): Asynchronous method for the synchronous doPost() method of OLAPService interface.

- **OLAPServiceImpl class:** Implements the OLAPService interface and enables access to the OLAP cloud.
doGet(): Get server information and virtual cube metadata from the OLAP cloud.
doDelete(): Delete specific virtual cube. **doPostMitHost():** Based on specific metadata create new virtual cube with certain number of associated hosts.
doPost(): Load WebRowSet XML data to a virtual cube.

- **Interface DataService:** RPC interface for the DataServiceImpl class.
getData()

- **Interface DataServiceAsync:** Asynchronous interface for interface DataService.
getData(): Asynchronous method for the synchronous getData() method of DataService interface.

- **Class DataServiceImpl:** Implements the DataService interface and enables access to Derby database.
getData(): Connect to Derby database, get raw data and transform it into WebRowSet XML format and return it to class Administrator.

- **Interface OgsaDaiService:** RPC interface for the OgsaDaiServiceImpl class.
getData()

- **Interface OgsaDaiServiceAsync:** Asynchronous interface for interface OgsaDaiService.
getData(): Asynchronous method for the synchronous getData() method of OgsaDaiService interface.

- **Class OgsaDaiServiceImpl:** Implements the OgsaDaiService interface and enables access to OGSA-DAI server.
getData(): Access OGSA-DAI server, get raw data and transform it into WebRowSet XML format and return it to class Administrator.

Chapter 4

Installation and Deployment

This chapter is a step by step guide, which introduces how to install the required software, how to compile the program and how to deploy the client systems. This chapter is the basis both for testing the client systems and for further development, so the development software tools are also covered.

4.1 Preparation

Following installation and deployment steps are all based on Windows XP operating system. Table 4.1 gives a list of all the software tools and libraries we applied for development.

Directory Structure

For the following part of the chapter let's assume we have a directory `c:\GWTOLAP` with following sub folders:

Name	Version	Download
Java	1.6.0	http://java.sun.com
Eclipse	3.6	http://www.eclipse.org
GWT	2.3.0	http://code.google.com/initl/de-DE/webtoolkit
GWT UI designer	for eclipse 3.6	http://code.google.com/initl/de-DE/webtoolkit
Ext GWT	2.2.4	http://www.sencha.com
Apache Wink	1.1.2	http://incubator.apache.org/wink
Derby	10.5.3	http://java.sun.com
Apache Ant	1.8.2	http://ant.apache.org
OGSA-DAI	4.0	http://sourceforge.net/projects/ogsa-dai
Apache Tomcat	7.0	http://tomcat.apache.org

Table 4.1: Applied software tools and libraries

```
c:\GWTOLAP\OLAPAdministrator
c:\GWTOLAP\OLAPQueryClient
c:\GWTOLAP\Database
c:\GWTOLAP\workspace
c:\GWTOLAP\apache-ant-1.8.2
c:\GWTOLAP\eclipse-java-helios-SR2-win32
c:\GWTOLAP\ogsadai-4.0-axis-1.4-bin
c:\GWTOLAP\wink
c:\GWTOLAP\apache-tomcat-6.0.33
c:\GWTOLAP\gxt-2.2.4
```

4.2 Installation

4.2.1 Installation for Both OLAP Query Client and OLAP Administrator

This section describes the installation and configuration steps for both client systems.

1. Java environment

Java environment is the only prerequisite for developing GWT web applications, at least Java Development Kit (JDK) 1.6 should be properly installed.

2. Eclipse

Eclipse is a popular Integrated Development Environment (IDE) for efficient software development. Besides, Google provides the GWT development plugin for Eclipse, so we decided to develop our application in Eclipse 3.6 (Helios).

3. GWT Eclipse Plugin and GWT Design

Besides the basic GWT development plugin, we also installed the GWT Design plugin which simplifies the GUI development by providing a series of existing reusable general components like buttons, tables etc. developer can utilize drag and drop functionality to accelerate the development progress. Following are the installation steps for the plugins.

In Eclipse mouse click: *Help* → *InstallNewSoftware*. In the "Install" window click "Add" button to add GWT plugin repository, according to the version of the Eclipse, we should enter the location: <http://dl.google.com/eclipse/plugin/3.6> , as shown in Figure 4.1.

For a different version of Eclipse, the corresponding GWT repository location should be given as shown in Table 4.2.

Click "OK" button, then we can get a list of available plugins as shown in Figure

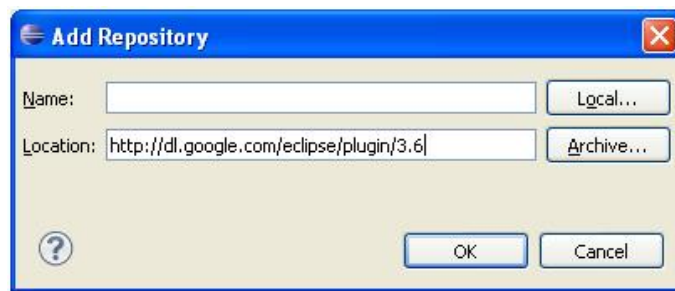


Figure 4.1: Add GWT plugin repository location

Version	GWT Plugin Location
Eclipse 3.7(In-digo)	http://dl.google.com/eclipse/plugin/3.7
Eclipse 3.6(Helios)	http://dl.google.com/eclipse/plugin/3.6
Eclipse 3.5(Galileo)	http://dl.google.com/eclipse/plugin/3.5
Eclipse 3.4(Ganymede)	http://dl.google.com/eclipse/plugin/3.4

Table 4.2: GWT repository locations for other version of Eclipse

4.2.

Select all the available items and click next, the progress information window will appear and show the downloading progress. Wait for download to finish, and the window in Figure 4.3 will appear.

Confirm the software package and click "next" button.(figure 4.4)

Start installing the software by selecting "I accept the terms of the license agreements" and clicking "Finish" button. Once installation complete, we should restart the Eclipse and a new Google toolbar will be available with three buttons: "New Web Application Project", "GWT Compile Project" and "Deploy App Engine Project", as shown in Figure 4.5.

4. Tomcat 6.0

Unpack the Tomcat archive file to:
c:\GWTOLAP\apache-tomcat-6.0.33

Define a new environment variable with the name CATALINA_HOME:
CATALINA_HOME = c:\GWTOLAP\apache-tomcat-6.0.33

To set environment variables open the control panel (Systemsteuerung) and look

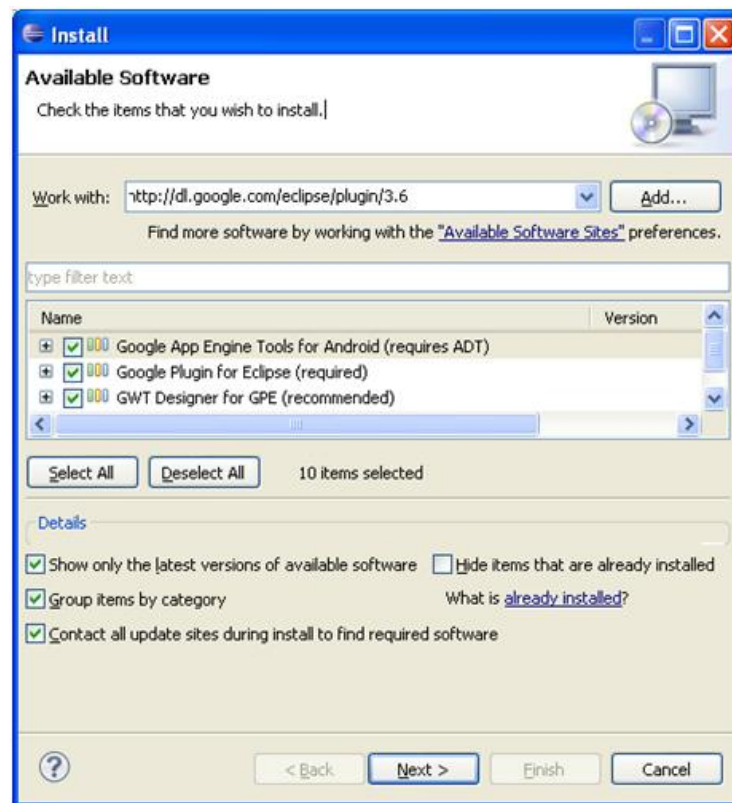


Figure 4.2: Plugins list from the repository

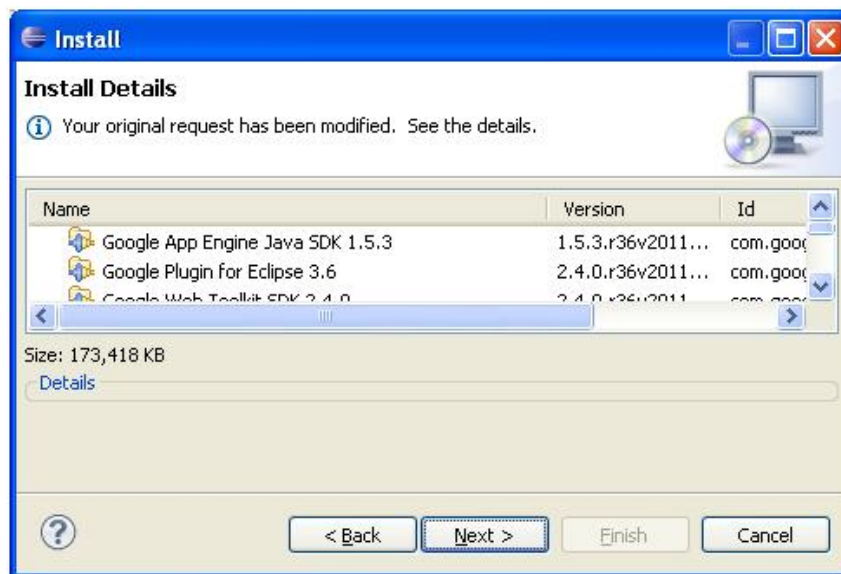


Figure 4.3: Confirm the installation

for system(System). Open the slider named advanced (Erweitert). Press the button environment variables(Umgebungsvariablen) and a window with a list of all existing variables will appear. (Figure 4.6).

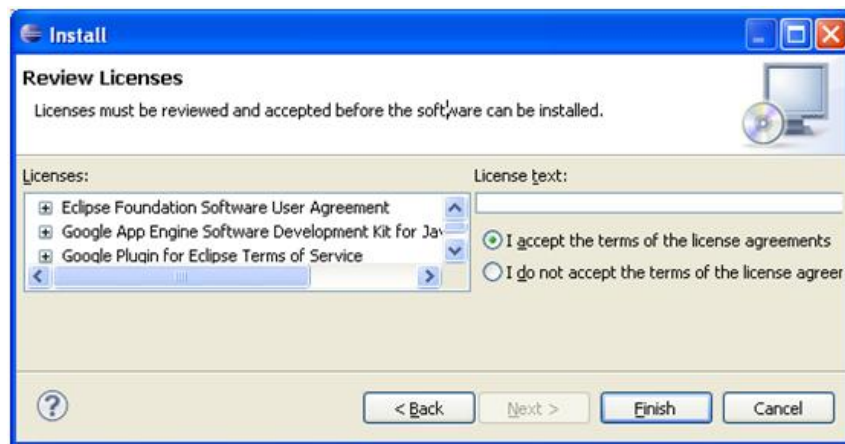


Figure 4.4: Accept the license agreement



Figure 4.5: The new Google toolbar

Start the Tomcat for the first time by using the startup script (startup.bat) located in:

```
c:\GWTOLAP\apache-tomcat-6.0.33\bin\startup.bat
```

If everything went fine the window as shown in Figure 4.7 will show up.

Stop the Tomcat by using another script (shutdown.bat) located in:

```
c:\GWTOLAP\apache-tomcat-6.0.33\bin\shutdown.bat
```

4.2.2 Installation for OLAP Query Client

This section focus on the installation of OLAP query client project in Eclipse as basis for extension and further development.

1. Import the OLAP Query Client project to Eclipse

In Eclipse, click *File* → *import*, then *General* → *Existing Project into Workspace* → *Next*.(Figure 4.8)

Click "Browse" button behind "Select root directory", then select the directory `c:\GWTOLAP\OLAPQueryClient`, mark the option "Copy projects into workspace" and then click "Finish" to complete the project import.

Now, in project explorer panel we can find source code under the folder "src". Yet, there should be still some red crosses in the source file `OLAPServiceImpl.java`,

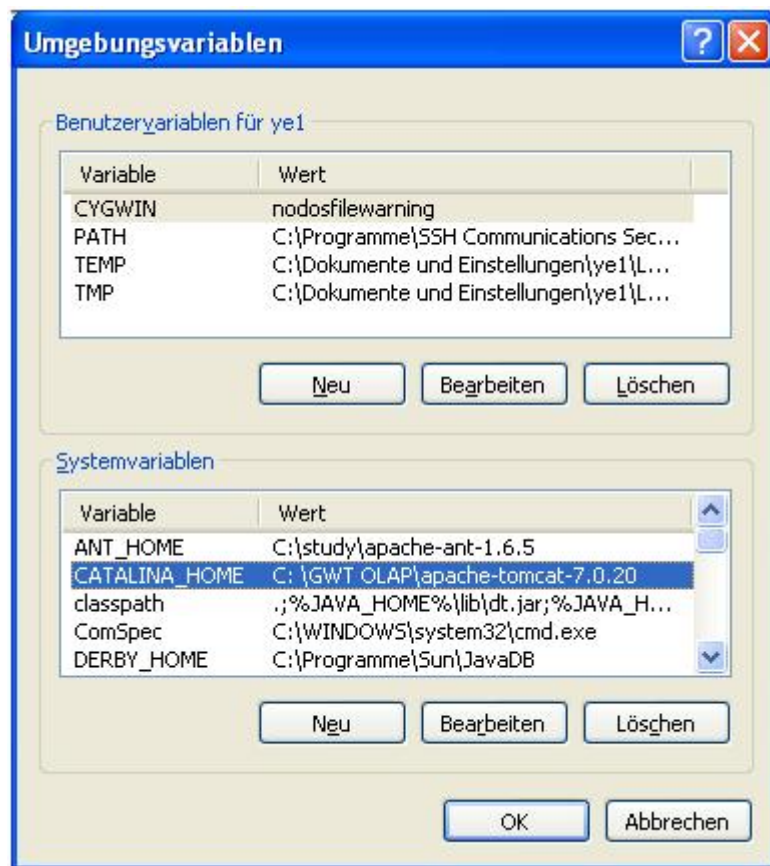


Figure 4.6: Environment variables

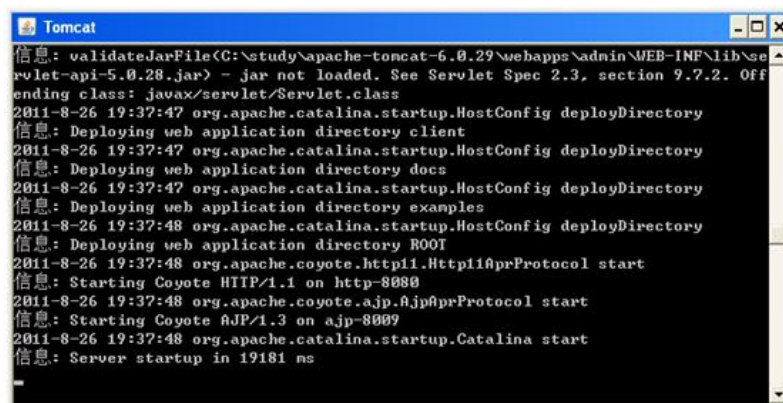


Figure 4.7: Start Tomcat 6.0

as this class has dependence on the Apache Wink library.

2. Add Apache Wink library to build path

Right click on the OLAPQueryClient project on the project explorer panel, then click *BuildPath* → *ConfigureBuildPath* to open the "Properties" window and select the "Library" tab (Figure 4.9). There should be a library "wink", select it and

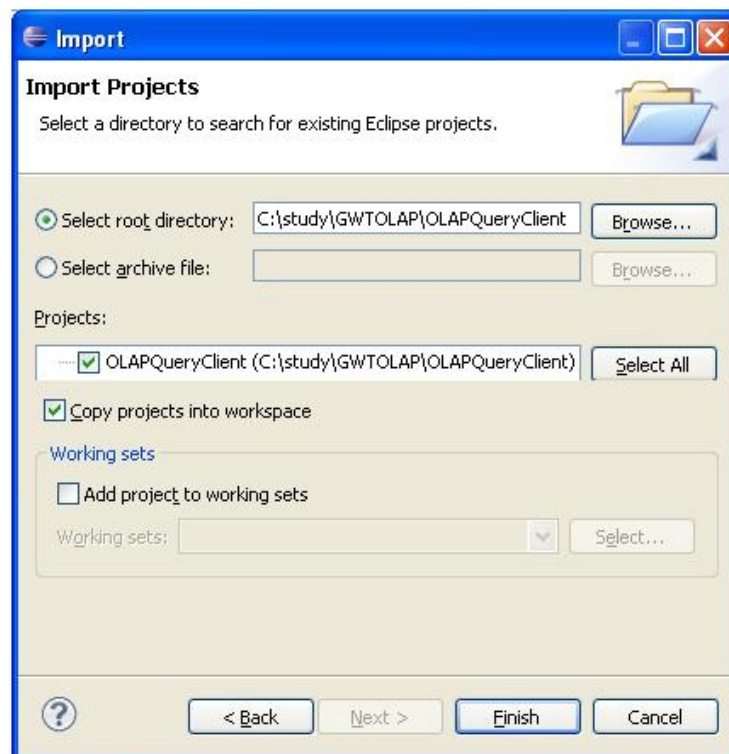


Figure 4.8: Import project window

click "Edit" button on the right side, click in "Edit Library" window *UserLibrary* → *NewUserLibrary*.

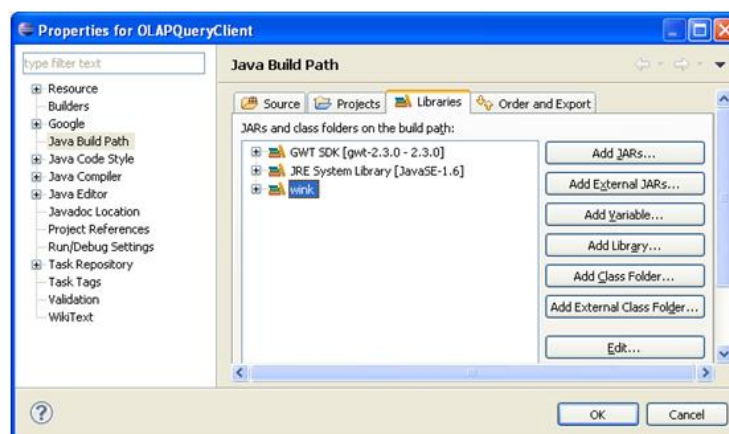


Figure 4.9: Project properties window

Enter "wink" in the "NewUserLibrary" window (Figure 4.10). Next, add the required Java ARchive (JAR) files to the new library. Then click "Add JARS" button, in the "JARSelection" window select jsr311-api-1.1.1.jar and wink-client-1.1.2-incubating.jar in c:\GWTOLAP\wink. Then click "OK" to finish (Figure 4.11).



Figure 4.10: Create new user library

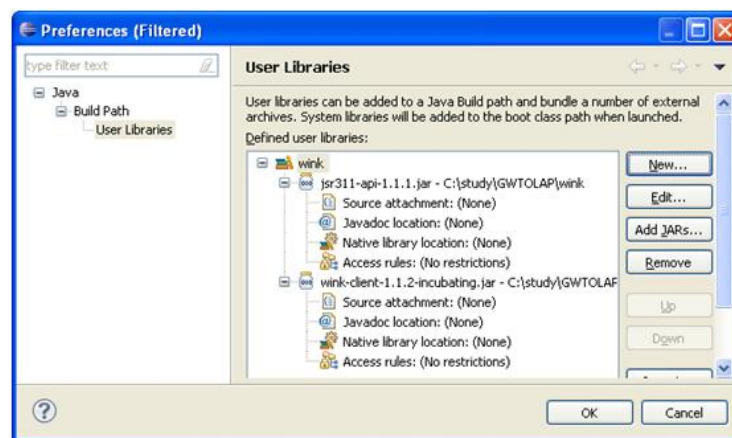


Figure 4.11: Add the required JAR files

3. Reallocate the GWT

As we have newly imported the project to Eclipse, we should also reallocate the path to our GWT environment. Analog to the last step, open the "Properties" window and edit the build path to GWT SDK. In the "Edit Library" window (Figure 4.12) mark "use specific SDK" and select GWT-2.3.0 and then click "finish" button, now, all the red crosses should be removed.

4. Open the GWT Design view to see the Layout

Navigate to the source code `OLAPQueryClient.java` and right click on it, by clicking *Openwith* → *GWTDesign* the GWT design view is opened (Figure 4.13). Besides, the tab "source" and "design" can help us to switch between the source code view and the design view.

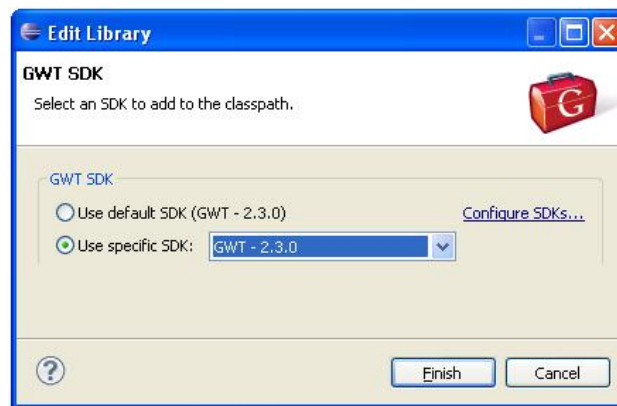


Figure 4.12: Edit library window

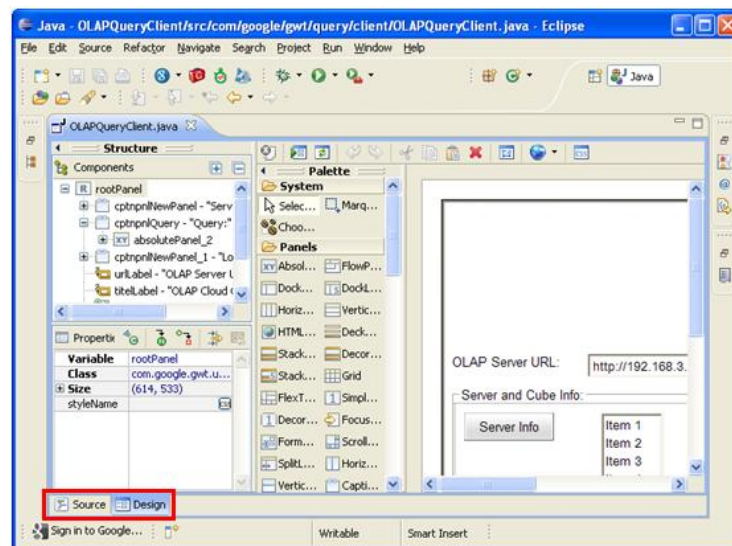


Figure 4.13: GWT Design view

4.2.3 Installation for OLAP Administrator

This section focus on the installation of OLAP administrator project in Eclipse as basis for extension and further development. Besides the ability to interact with the OLAP cloud like the query client does, the OLAP administrator should also interact with our data sources including Derby database and OGSA-DAI server, so more steps are needed.

1. Derby DB

Java DB is Sun's supported distribution of the open source Apache Derby DB database [Der], Sun bundles it with their JDK, and this JDK is the easiest way to install Java DB on Microsoft Windows. As we want to use Java DB to provide databases for our applications, we should select the Java DB component from the list of options when we install the Sun JDK.

To use Java DB, add the directory for the Java DB executable files to environment variable PATH: `c:\Program Files\Sun\JavaDB\bin`

Java DB also requires a separate DERBY_HOME environment variable that points to the root directory for the Java DB installation: `c:\Program Files\Sun\JavaDB`

To test the Java DB installation, open a terminal (command line) window, and type: `Sysinfo` (Figure 4.14).

```

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Dokumente und Einstellungen\ye1>sysinfo
----- Java Information -----
Java Version:      1.6.0_23
Java Vendor:      Sun Microsystems Inc.
Java home:        C:\Programme\Java\jdk1.6.0_23\jre
Java classpath:   .;C:\Programme\Java\jdk1.6.0_23\lib\dt.jar;C:\Programme\Java\jdk1.6.0_23\lib\tools.jar;C:\study\us-core-4.0.3\lib\bootstrap.jar;C:\cygwin\home\ye1\src\org.gridniner.jar;C:\Programme\Sun\JavaDB\lib\derbyclient.jar;C:\Programme\Sun\JavaDB\lib\derby.jar;C:\study\mysql-connector-java-5.1.16\mysql-connector-java-5.1.16-bin.jar;c:\tools\igoux.jar;C:\Programme\Sun\JavaDB\lib\derby.jar;C:\Programme\Sun\JavaDB\lib\derbynet.jar;C:\Programme\Sun\JavaDB\lib\derbyclient.jar;C:\Programme\Sun\JavaDB\lib\derbytools.jar
OS name:          Windows XP
OS architecture: x86
OS version:       5.1
Java user name:   ye1
Java user home:   C:\Dokumente und Einstellungen\ye1
Java user dir:    C:\Dokumente und Einstellungen\ye1
java.specification.name: Java Platform API Specification
java.specification.version: 1.6
----- Derby Information -----
JRE - JDBC: Java SE 6 - JDBC 4.0

```

Figure 4.14: Sysinfo from Derby

2. Apache Ant

Unpack the Apache Ant archive file to:
`c:\GWTOLAP\apache-ant-1.8.2`

Define a new environment variable named ANT_HOME:
`ANT_HOME = C:\GWTOLAP\apache-ant-1.8.2`
 Extend the PATH variable with `%ANT_HOME%\bin`

3. OGSA-DAI

Unpack the Apache Ant archive file to:
`c:\GWTOLAP\ogsadai-4.0-axis-1.4-bin`

Define a new environment variable named OGSADAI_HOME:
`OGSADAI_HOME = c:\GWTOLAP\ogsadai-4.0-axis-1.4-bin`

To set the CLASSPATH in an OGSA-DAI binary distribution:
`cd %OGSADAI_HOME%`

setenv.bat

put the database driver JARs (derby.jar and derbyclient.jar) into the following directory:

OGSADAI_HOME\thirdparty\lib

Extend the CLASSPATH variable with:

OGSADAI_HOME\thirdparty\lib\derby.jar

OGSADAI_HOME\thirdparty\lib\derbyclient.jar

Deploy OGSA-DAI Axis to Tomcat (Figure 4.15):

cd %OGSADAI_HOME%

ant -Dtomcat.dir=%CATALINA_HOME% buildAndDeployWAR

```

-INF\etc\dai
[copy] Copying 17 files to C:\study\GWTOLAP\ogsadai-4.0-axis-1.4-bin\build\
war\MEB-INF\etc\dai
[copy] Copied 5 empty directories to 2 empty directories under C:\study\GWT
OLAP\ogsadai-4.0-axis-1.4-bin\build\war\MEB-INF\etc\dai
[copy] Copying 1 file to C:\study\GWTOLAP\ogsadai-4.0-axis-1.4-bin\build\wa
r\MEB-INF\classes
[war] Building war: C:\study\GWTOLAP\ogsadai-4.0-axis-1.4-bin\build\dai.wa
r
[echo] Done!

deployWAR:
[echo] Deploying webapp to Tomcat...
[copy] Copying 1 file to C:\study\GWTOLAP\apache-tomcat-7.0.20\webapps
[mkdir] Created dir: C:\study\GWTOLAP\apache-tomcat-7.0.20\webapps\dai
[copy] Copying 149 files to C:\study\GWTOLAP\apache-tomcat-7.0.20\webapps\d
ai
[copy] Copied 30 empty directories to 2 empty directories under C:\study\GW
TOLAP\apache-tomcat-7.0.20\webapps\dai
[echo] Done!

BUILD SUCCESSFUL
Total time: 12 seconds
C:\study\GWTOLAP\ogsadai-4.0-axis-1.4-bin>

```

Figure 4.15: Deploy OGSA-DAI Axis onto Tomcat

Start the Tomcat by running the startup script (startup.bat) located in:
c:\GWTOLAP\apache-tomcat-6.0.33\bin\startup.bat

Visit the following page using an web page browser:

<http://localhost:8080/dai/services>

If the services list of the OGSA-DAI server shows up, the installation is successful.

4. Import the OLAP administrator project

From c:\GWTOLAP\OLAPAdministrator to Eclipse (see Section 4.2.2 step 1)

5. Add Apache Wink library to build path (see Section 4.2.2 step 2)

6. Reallocate the GWT (see Section 4.2.2 step 3)

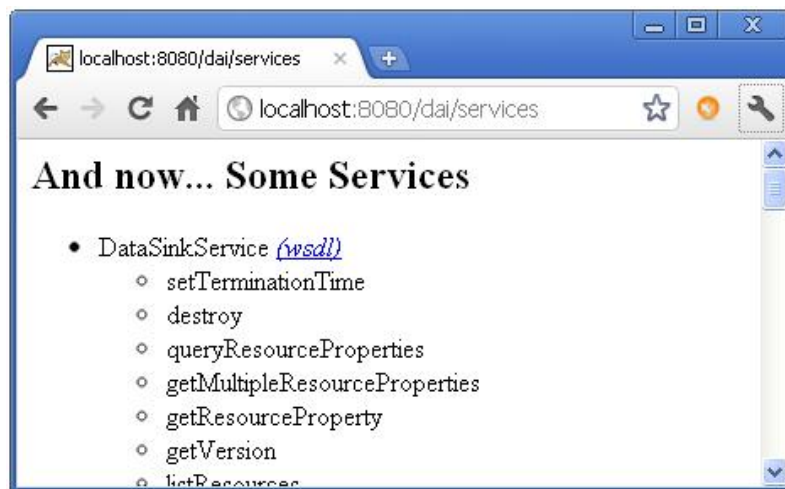


Figure 4.16: OGSA-DAI: deployed services list

7. Add OGSA-DAI library to build path

Similar to adding Apache Wink's library to the build path, we should import following JAR files from

`c:\GWTOLAP\ogsadai-4.0-axis-1.4-bin\lib` to the OLAP administrator project:

- `ogsadai-4.0-axis-1.4-client.jar`
- `ogsadai-4.0-axis-1.4-clientserver.jar`
- `ogsadai-4.0-axis-1.4-extensions-client.jar`
- `ogsadai-4.0-axis-1.4-common.jar`

8. Open the GWT Design view to see the Layout (see Section 4.2.2 step 4)

9. install and configure Ext GWT

First, the file `gxt.jar` from `c:\GWTOLAP\gxt-2.2.4` should be added as external JAR to the Java build path. Then, copy all files of directory `c:\GWTOLAP\gxt-2.2.4\` to the project's `war/gxt` directory (Figure 4.17).

There should be a warning:

"The following classpath entry `c:\GWTOLAP\gxt-2.2.4\gxt.jar` will not be available on the server's classpath",

which indicates the `gxt.jar` is in the Java build path for development but not in the runtime classpath (`WEB-INF/lib`). As `gxt.jar` is needed only at compile time, we can remove the warning simply by opening the project properties window, selecting Google-Web Application item and adding `gxt.jar` to

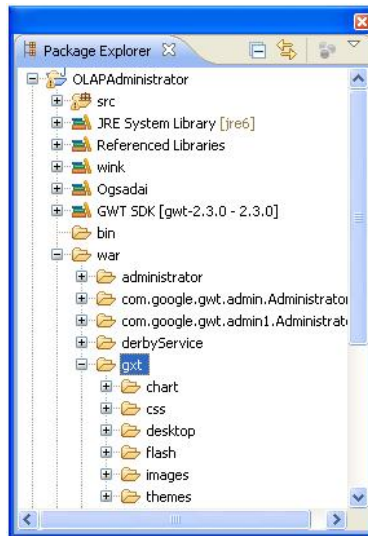


Figure 4.17: Directory structure of the OLAP administrator project

Suppress warning about these build path entries being outside of WEB/lib.

Additionally, Ext GWT should be added to the module definition file of the project. We need to open the file `Administrator.gwt.xml`, add the module `com.extjs.gxt.ui.GXT` to module inheritance and add the `gxt-all.css` to the module:

```
<inherits name="com.extjs.gxt.ui.GXT"/>
<stylesheet src="../gxt/css/gxt-all.css"/>
```

4.3 Deployment

4.3.1 Deployment for OLAP Query Client

1. If there is any changes on the source code or design layout of the project, the project should be compiled again: click in Google toolbar, *GWTCompileProject* → *Compile*.
2. The compile result will be in the "war" directory of the project.
3. We can simply copy the "war" directory to the web application deployment folder of Apache Tomcat: `c:\GWTOLAP\apache-tomcat-6.0.33\webapps`
4. Change the directory's name from "war" to "client".
5. Start Apache Tomcat using the script:
`c:\GWTOLAP\apache-tomcat-6.0.33\bin\startup.bat`
6. Visit `http://localhost:8080/client` in a web browser, the GUI of the OLAP query client appears as shown in Figure 4.18.



Figure 4.18: Initial appearance of the OLAP query client GUI

4.3.2 Deployment for OLAP Administrator

Besides deploy the OLAP Administrator project to Apache Tomcat application server, in this section there is also description about deployment of the Derby database and OGSA-DAI resources based on the data set which is described in Chapter 5. So after the following deployment steps, the OLAP administrator is ready for a fully functional test.

1. Create databases and tables in Derby.

Start Derby network server:

```
cd c : \GWTOLAP\Database
```

```
startNetwaorkServer -h 0.0.0.0 -p 1527 -noSecurityManager
```

Here, the option `-h 0.0.0.0` means that client is not restricted by the IP address to access to the server.

Create new databases using the "ij" tool from Derby, start another command line window and run:

```
ij
```

```
CONNECT 'jdbc:derby://host:1527/DatabaseName;create=true';
```

Here, "host" is the Derby network server IP

Connect to an existing Database, like in `c:\GWTOLAP\Database` there are two

existing databases forestFireDB1 and forestFireDB2.

```
CONNECT 'jdbc:derby://host:1527/databaseName';
```

Create new table in the new database:

```
CREATE TABLE tableName(
RecordID varchar(50) NOT NULL default '',
X varchar(50) NOT NULL default '',
Y varchar(50) NOT NULL default '',
Month varchar(5) NOT NULL default '',
Day varchar(5) NOT NULL default '',
FFMC double NOT NULL default 0,
DMC double NOT NULL default 0,
DC double NOT NULL default 0,
ISI double NOT NULL default 0
);
```

Insert record into the table:

```
INSERT INTO tableName VALUES ('1','2','3','10','2',12,10,1,98);
```

2. Deploy database resources to OGSA-DAI.

In folder `c:\GWTOLAP\ogsadai-4.0-axis-1.4-bin` create OGSA-DAI configuration file with following contents:

```
DeployResource deployJDBC MySQLResource jdbc:derby://
host:1527/database org.apache.derby.jdbc.ClientDriver
Login permit MySQLResource ANY myUser somePassword
```

For example, configuration file "fpart1" is used to configure the database "forestFireDB1" as resource "FFDB1" on OGSA-DAI.

```
DeployResource deployJDBC FFDB1 jdbc:derby://
192.168.1.101:1527/forestFireDB1 org.apache.derby.jdbc
.ClientDriver Login permit FFDB1 ANY APP ANY
```

Configuration file "fpart1" is used to configure the database "forestFireDB2" as resource "FFDB2" on OGSA-DAI.

```
DeployResource deployJDBC FFDB2 jdbc:derby://
192.168.1.101:1527/forestFireDB2 org.apache.derby.jdbc
.ClientDriver Login permit FFDB2 ANY APP ANY
```

Deploy the configuration files, start a new command line window:

```
cd %OGSADAI_HOME%
ant -Dtomcat.dir =%CATALINA_HOME% -Dconfig.file =
CONFIG - FILE [-Djar.dir = JAR - DIRECTORY] [-Dstart.line = LINE]
configure
```

For example, deploy configuration file "fpart1":

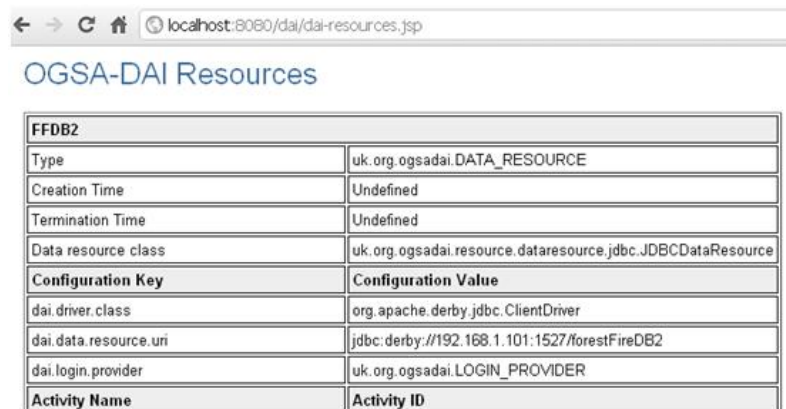
```
ant -Dtomcat.dir =%CATALINA_HOME%-Dconfig.file = fpart1 configure
```

deploy configuration file fpart2:

```
ant -Dtomcat.dir =%CATALINA_HOME%-Dconfig.file = fpart2 configure
```

Restart Tomcat and the deployed resources should appear in:

<http://localhost:8080/dai/dai-resources.jsp> (Figure 4.19)



FFDB2	
Type	uk.org.ogsadai.DATA_RESOURCE
Creation Time	Undefined
Termination Time	Undefined
Data resource class	uk.org.ogsadai.resource.dataresource.jdbc.JDBCDataResource
Configuration Key	Configuration Value
dai.driver.class	org.apache.derby.jdbc.ClientDriver
dai.data.resource.uri	jdbc:derby://192.168.1.101:1527/forestFireDB2
dai.login.provider	uk.org.ogsadai.LOGIN_PROVIDER
Activity Name	Activity ID

Figure 4.19: Deployed resources on OGSA-DAI server

3. Configure and deploy OGSA-DAI DQP resources.

Again in folder `c:\GWTOLAP\ogsadai-4.0-axis-1.4-bin`

Create a DQP resource configuration file: "dqp_resource.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
<DQPResourceConfig>
  <dataResources>
    <resource url="http://localhost:8080/dai/services"
      resourceID="FFDB1"
      isLocal="true"/>
    <resource url="http://localhost:8080/dai/services"
      resourceID="FFDB2"
      isLocal="true" />
  </dataResources>
</DQPResourceConfig>
```

Deploy a DQP resource:

create a file `conf.dqp` with the content:

```
DeployResource deployDQP RESOURCE_ID FILE
```

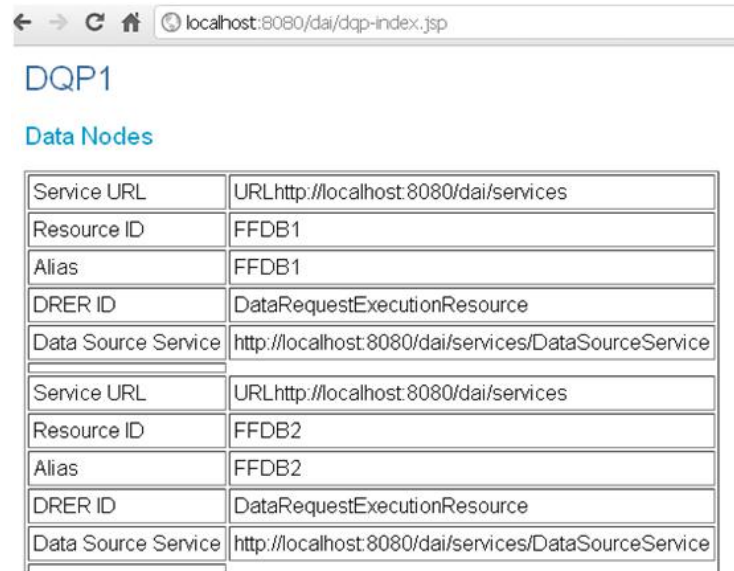
For example:

```
DeployResource deployDQP DQP1 dqp_resource.xml
```

Deploy the configuration file, start a new command line window:

```
cd %OGSADAI_HOME%
ant -Dtomcat.dir=%CATALINA_HOME%-Dconfig.file=conf_dqp configure
```

Restart Tomcat and the deployed DQP resource should appear in:
<http://localhost:8080/dai/dqp-index.jsp> (Figure 4.20)



Service URL	URLhttp://localhost:8080/dai/services
Resource ID	FFDB1
Alias	FFDB1
DRER ID	DataRequestExecutionResource
Data Source Service	http://localhost:8080/dai/services/DataSourceService
Service URL	URLhttp://localhost:8080/dai/services
Resource ID	FFDB2
Alias	FFDB2
DRER ID	DataRequestExecutionResource
Data Source Service	http://localhost:8080/dai/services/DataSourceService

Figure 4.20: DQP resource on OGSA-DAI server

4. Click "GWT Compile Project" button in Google toolbar to open the "GWT Compile" window, keep all default settings and click "Compile" to compile the project.
5. The compile result will be in the "war" directory of the project.
6. Copy the "war" directory to the folder:
c:\GWTOLAP\apache-tomcat-6.0.33\webapps
7. Change the name of the directory from "war" to "admin".
8. Restart Derby DB server. `cd c:\GWTOLAP\Database`
`startNetworkServer -h 0.0.0.0 -p 1527 -noSecurityManager`
9. Restart Apache Tomcat using the script:
c:\GWTOLAP\apache-tomcat-6.0.33\bin\startup.at
10. Visit <http://localhost:8080/admin> in a web browser, the GUI of the OLAP administrator appears as shown in Figure 4.21.



Figure 4.21: Initial appearance of the OLAP administrator GUI

Chapter 5

Graphical User Interface

There are two Graphical User Interfaces (GUI) in my implementation. The OLAP query client GUI is designed for query user who has right to access and query the virtual cube server in the OLAP cloud and load one row of data to a virtual cube. The OLAP administrator GUI is designed for administrator user who has right to query virtual cube server, create new virtual cubes, delete virtual cubes and also load large amount of data to a virtual cube in a single submission. Besides, the OLAP administrator GUI also enables user to load data from two different data sources including Derby database and OGSA-DAI server, further the data can be forwarded to OLAP cloud.

In this chapter, let's first introduce a data set which is suitable for OLAP analysis. Then, based on this data set we introduce step-by-step usage of the two GUIs.

5.1 Description of Testing Data Set

From the UCI machine learning repository [UCI] we can find various kinds of data sets which are not only suitable for data mining [HK00] but also for other kinds of knowledge discovery analyses. The testing data set here is the ForestFire data set from UCI machine learning repository, which is available at:
<http://archive.ics.uci.edu/ml/datasets/Forest+Fires>

The data set has following features:

- The data set contains data records of forest fires in Montesinho natural park in Portugal.
- The data records were collected between 2000 and 2003 by two observers:
 - Inspectors who are responsible for the Montesinho forest fire, they contribute 8 attributes to the data set.
 - Braganca Polytechnic Institute, they contribute 5 attributes to the data set.
- The data set contains 517 records with each 13 attributes.

Following is the description of the attributes.

- Attributes from the first observer's database:
 - X - x-axis location description: 1 to 9
 - Y - y-axis location description: 2 to 9
 - month - month in year : 'jan' to 'dec'
 - day - weekdays: 'mon' to 'sun'
 - FFMC - FFMC index from FWI system: 18.7 to 96.20
 - DMC - DMC index from FWI system: 1.1 to 291.3
 - DC - DC index from FWI system: 7.9 to 860.6
 - ISI - ISI index from FWI system: 0.0 to 56.10
- Attributes from the second observer's database:
 - temp - Temperature in Celsius degrees: 2.2 to 33.30
 - RH - relative humidity in percentage: 15.0 to 100
 - wind - wind speed in km/h: 0.40 to 9.40
 - rain - rain in mm/m2 : 0.0 to 6.4
 - area - burned area of the forest (in ha): 0.00 to 1090.84

5.1.1 Prerequisites

For the following functional usage description of the GUIs, we prepare two Derby databases (forestFireDB1 and forestFireDB2) which contains each the database records from a observer of the forest fire data set. Further, we establish a OGSA-DAI server and deploy the two databases as two relational data resources (FFDB1 and FFDB2) to it and deploy a Distributed Query Processing (DQP) resource (DQP1), and associate FFDB1 and FFDB2 to the DQP resource. Details on how to deploy the databases and OGSA-DAI server can be found in Chapter 4.

5.2 Introduction of OLAP Administrator GUI

Figure 5.1 presents the OLAP administrator GUI, in the following we walk through a step-by-step example to show the usage.

1. Choose the virtual cube server from the dropdown menu. New virtual cube server URI can be added by click the "Add URL" button.
2. There are two ways to get the raw data:
first, Directly from Derby database.

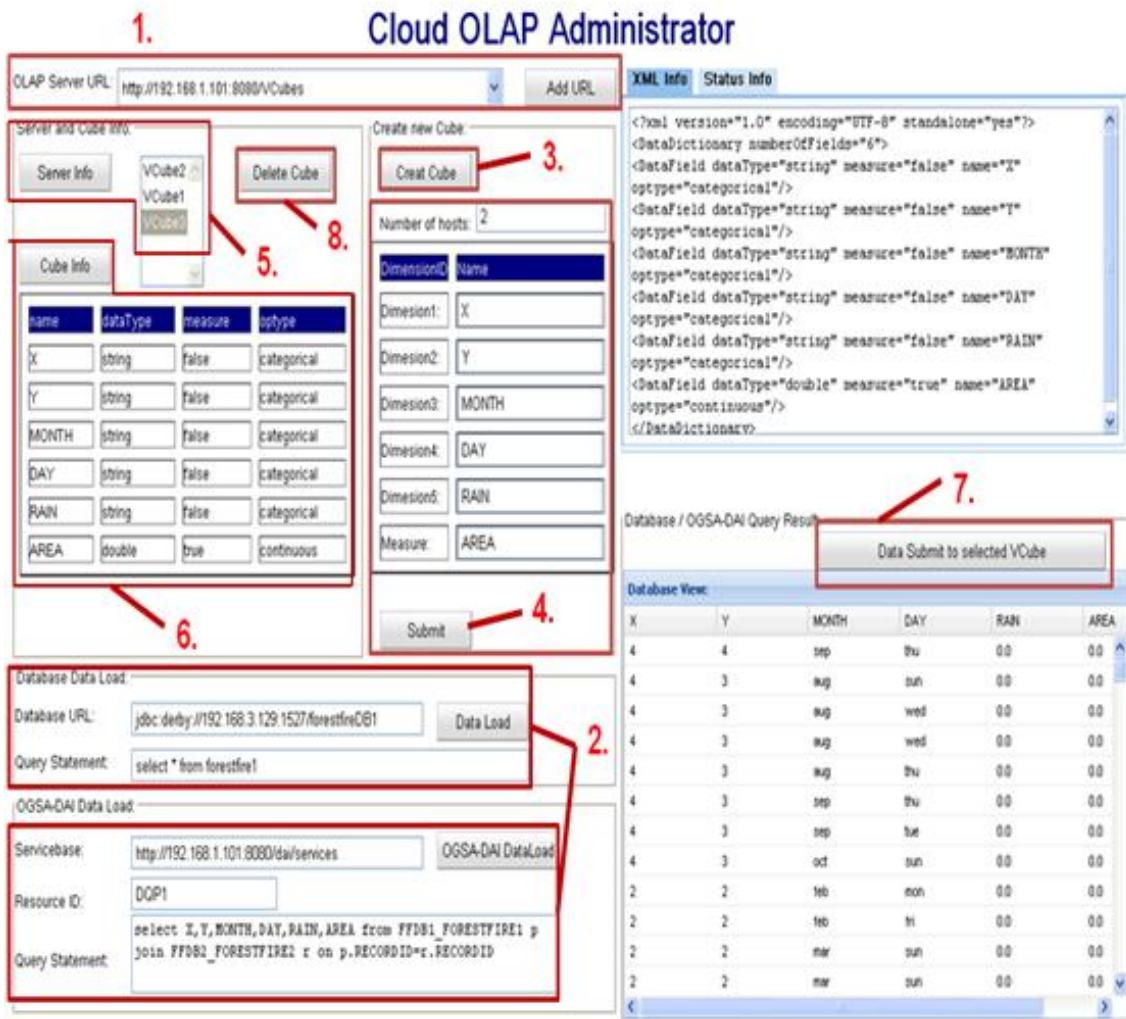
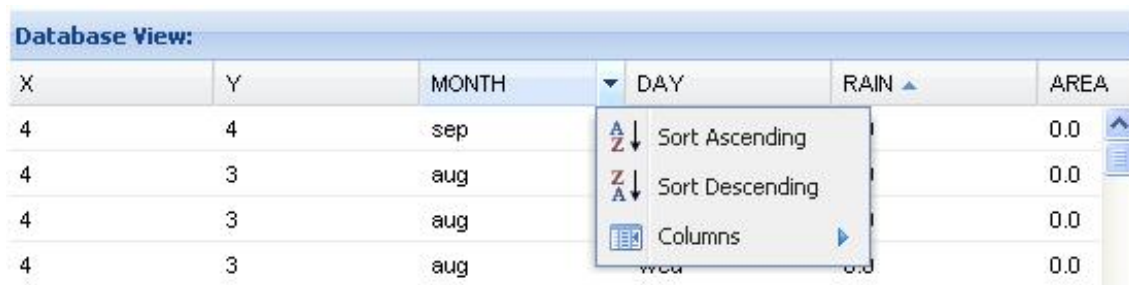


Figure 5.1: OLAP administrator GUI

- (a) Enter the database URL in textbox right to "Database URL":
jdbc:derby://host:port/databaseName
- (b) Enter the query statement in textbox right to "Query statement":
select * from forestfire1
- (c) click "Data Load" button, the result will appear in "Database/OGSA-DAI Query Result" panel as Ext GWT grid which can be easily sorted or edited (figure 5.2). The resulted data in WebRowSet XML format will also be shown in "XML state" panel.



X	Y	MONTH	DAY	RAIN	AREA
4	4	sep			0.0
4	3	aug			0.0
4	3	aug			0.0
4	3	aug			0.0

Figure 5.2: Ext GWT grid

Second, get data from OGSA-DAI server.

- (a) Enter the OGSA-DAI server URL in textbox right to "Service Base":
- (b) Enter the Resource ID "DQP1"
- (c) In the textbox right to "Query Statement" enter:
" select X, Y, MONTH, DAY, RAIN, AREA from FFDB1_FORESTFIRE1
p join FFDB2_FORESTFIRE2 r on p.RECORDID=r.RECORDID".
The result set of the query should contains columns from two databases
and these columns should be integrated to form a result set.
- (d) click "OGSA-DAI Data Load" button, the result will appear in
"Database/OGSA-DAI Query Result" panel as Ext GWT grid. The re-
sulted data in WebRowSet XML format will also be shown in "XML
state" panel.

Following, assume we have load data from OGSA-DAI server using the above statement.

3. Click "Create Cube" button and enter the number of dimensions of the new virtual cube and dimension names exactly the same as what we have loaded from the OGSA-DAI server.
4. enter in the textbox right to "Number of hosts" the number of hosts used by the virtual cube, then click "Submit" button. Once the new virtual cube is created we will get a result indicating the URI of the virtual cube in "state" panel.

5. Click "Server Info" button to get the list of available virtual cube on the server, the newly created virtual cube should be in the list, assume its cubeID is "VCube3".
6. Select "VCube3" and click "Cube Info" button, metadata of the cube should show up as a table. The <DataDictionary> will be shown in "XML State" panel.
7. Click "Data Submit to selected VCube" button to load the data from the Ext GWT grid to the virtual cube. If succeed, result information will appear in "state" panel as follows:"517 row loaded, cube construction time: 3813 ms, response time: : 4264 ms"
8. By selecting an available virtual cube and click the "Delete Cube" button, we can remove a virtual cube on the server.

5.3 Introduction of OLAP Query Client GUI

Figure 5.3 presents the OLAP query client GUI, again, we walk through a step-by-step example to show the usage.

1. Choose the virtual cube server from the dropdown menu. New virtual cube server URI can be added by click the "Add URL" button.
2. Click "Server Info" button to get the list of available virtual cube on the server.
3. Select "VCube3" (created in the section 5.2) and click "Cube Info" button, metadata of the cube should show up as a table. The <DataDictionary> will be shown in "XML State" panel.
4. To initiate an OLAP query, first select the function (POINT, SUM, MIN or MAX) from the dropdown list and click "create query table" button. A query table according to the selected virtual cube's metadata will be created. In the query table select values for each dimension, e.g. X = 4, Y = 3, MONTH = [ANY], DAY = [ANY], RAIN = 0.0 and select the SUM function. As we have the burned forest area as our measure, such a query will answer the question: "In location (X=4, Y=3), at any time, when there is no rain, how much area was burned in total?"
5. Click "Query" button, the query will be submitted, if succeed, the answer should be presented right after the query table: "AREA= 167.18" which indicates totally 167.18 ha of the forest was burned when we have above conditions.
6. We can also load one row of data from the OLAP query client GUI. Click "create Loat Table" button, enter the values for each dimension and measure. Here, value for dimension should not be empty, and value for measure should be a double precision floating point number.

- Click "load data" button, the one row of data will be loaded to the virtual cube. This step can also be repeated when more rows of data should be loaded.

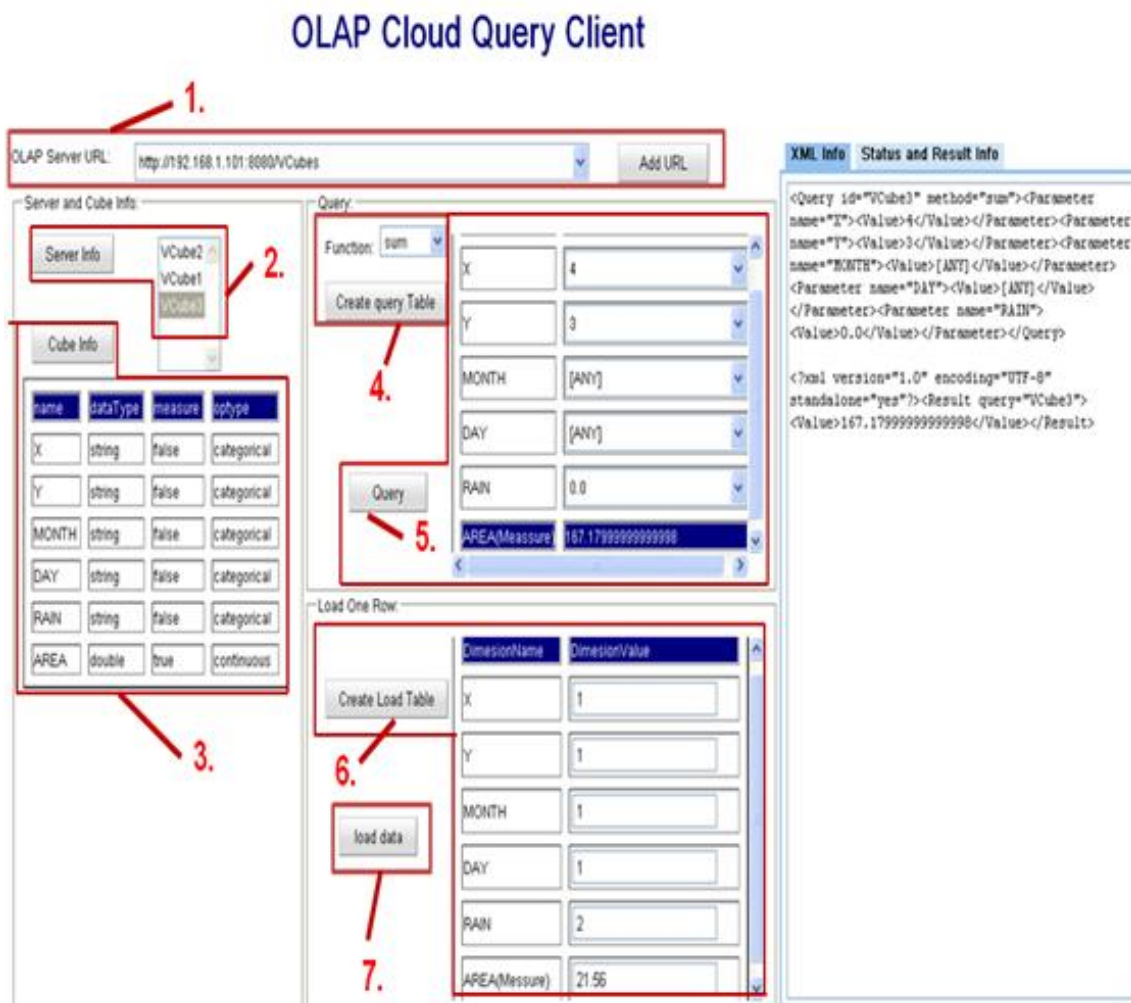


Figure 5.3: OLAP query client GUI

Chapter 6

OLAP Modelling Markup Language

Typically, each OLAP system uses its own proprietary data cube specification language which is understandable only by its own OLAP platform. In order to guarantee a sufficient level of interoperability - the ability of two or more systems to exchange and use information, among different OLAP system, the OLAP Model Markup Language (OMML), is designed to describe consistent OLAP models and it is independent of any target OLAP platforms. This standard representation can be used by other OMML compatible applications for further processing of the same data cube, for example data mining tools or graphical representation of OLAP models.

OMML was first introduced in [EO05]. We have further develop the first version of OMML with modifications and extensions, which resulted in OMML version 2.0. OMML 2.0 will be described in this chapter, It is designed to be suitable for representation of OLAP queries and results.

6.1 The Components of OMML

The root tag of the OMML XML Schema Definition (XSD) is <OMML>, which has five components: <Header> contains general information of OMML, <ServerInfo> contains virtual cube server information, <DataDictionary> contains metadata of a virtual cube, <Query> describes the OLAP query and <Result> represents the OLAP query result.

```
<xs:element name="OMML">
  <xs:complexType>
    <xs:annotation>
      <xs:documentation>OLAP Model Markup Language</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element ref="Header"/>
      <xs:element ref="ServerInfo"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```

    <xs:element ref="DataDictionary"/>
    <xs:element ref="Query" maxOccurs="unbounded" />
    <xs:element ref="Result" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="version" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>

```

6.2 General Information

Within the <Header> tag, there are three elements: <TimeStamp> contains the OMML document creation time, <ServiceURI> contains service URI of the server and <Annotation> can includes some annotation information. There are also two attribute here: copyright and description.

```

<xs:element name="Header">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Annotation" minOccurs="0"/>
      <xs:element ref="Timestamp" minOccurs="0"/>
      <xs:element ref="ServiceURI" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="copyright" type="xs:string" use="required"/>
    <xs:attribute name="description" type="xs:string"/>
  </xs:complexType>
</xs:element>
<xs:element name="Annotation" type="xs:string"/>
<xs:element name="Timestamp" type="xs:string"/>
<xs:element name="ServiceURI" type="xs:string"/>

```

6.3 Virtual Cube Server Information

The second tag <ServerInfo> contains the list of available virtual cubes on a virtual cube server. The cubeIDs are listed one after another.

```

<xs:element name="ServerInfo">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CubeID" type="xs:string"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Following is example of a <ServerInfo> XML document which contains IDs of three virtual cubes. URI of a virtual cube is the server's URI plus the virtual cube ID, e.g. <http://{serverIP}:{port}/VCubes/VCube1>.


```

<ServerInfo>
  <CubeID>VCube2</CubeID>
  <CubeID>VCube1</CubeID>
  <CubeID>VCube3</CubeID>
</ServerInfo>

```

6.4 Metadata and Dimension Hierarchies of Virtual Cube

<DataDictionary> contains the metadata and dimension hierarchy information of a virtual cube. The attribute "numberOfFields" indicates how many data fields the virtual cube has, i.e. number of dimensions plus measure.

```

<xs:element name="DataDictionary">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="DataField" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="numberOfFields" type="xs:integer"
      use="required"/>
  </xs:complexType>
</xs:element>

```

Every <DataField> describes a dimension or measure. Attribute "name" represents the name. Attribute "optype" describes the relationship between the dimension members, it could be categorical, ordinal or continuous. Attribute "dataType" represents the dimension's data type which contains eight possible types and can be extended. Attribute "measure" is used to indicate if the data field is measure or a dimension. Attribute "numberOfHierarchyLevels" indicates the number of hierarchical levels of a dimension. Attribute "hierarchyLevel" represents the current level of the dimension. The two attributes about the dimension hierarchy is unrequired, as there could be no hierarchical levels for a dimension. The Element <value> describes all the dimension members of the dimension, if the dimension has different hierarchical levels, it describes all dimension members of all levels.

```

<xs:element name="DataField">
  <xs:complexType >
    <xs:sequence>
      <xs:element name="Value" type="xs:string" minOccurs="0"
        maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="optype" type="OPTYPE" use="required"/>
    <xs:attribute name="dataType" type="DATATYPE" use="required"/>
    <xs:attribute name="measure" type="xs:boolean" use="required"/>
    <xs:attribute name="numberOfHierarchyLevels" type="xs:integer"

```

```

        use="unrequired"/>
        <xs:attribute name="hierarchyLevel" type="xs:integer"
            use="unrequired"/>
    </xs:complexType>
</xs:element>
<xs:simpleType name="OPTYPE">
    <xs:restriction base="xs:string">
        <xs:enumeration value="categorical"/>
        <xs:enumeration value="ordinal"/>
        <xs:enumeration value="continuous"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="DATATYPE">
    <xs:restriction base="xs:string">
        <xs:enumeration value="string"/>
        <xs:enumeration value="integer"/>
        <xs:enumeration value="float"/>
        <xs:enumeration value="double"/>
        <xs:enumeration value="boolean"/>
        <xs:enumeration value="date"/>
        <xs:enumeration value="time"/>
        <xs:enumeration value="dateTime"/>
    </xs:restriction>
</xs:simpleType>

```

The following example describes a cube whose dimensions have no different hierarchical levels. The cube has three dimensions, "Location", "Year", "Product", and a measure, "sale". Each dimension member is in a <value> element.

```

<DataDictionary numberOfFields="4">
    <DataField name="Location" optype="categorical"
        dataType="string" measure="false" >
        <Value>Austria</Value>
        <Value>USA</Value>
        <Value>UK</Value>
    </DataField>
    <DataField name="Year" optype="categorical"
        dataType="integer" measure="false" >
        <Value>2011</Value>
        <Value>2010</Value>
        <Value>2009</Value>
    </DataField>
    <DataField name="Product" optype="categorical"
        dataType="string" measure="false" >
        <Value>PC</Value>
        <Value>TV</Value>
        <Value>DVD</Value>

```

```

</DataField>
<DataField name="Sale" optype="continuous"
  dataType="double" measure="true"/>
</DataDictionary>

```

If the first dimension "Location" from above example has four hierarchical levels, the related <DataField> should be described as below. Here, each <Value> describe a dimension member on a hierarchical level.

```

<DataField name="Location" optype="categorical"
  dataType="string" measure="false"
  numberOfHierarchyLevels="4" hierarchyLevel="3">
  <Value>Europe</Value>
  <Value>North America</Value>
  <Value>Asia</Value>
</DataField>
<DataField name="Location" optype="categorical"
  dataType="string" measure="false"
  numberOfHierarchyLevels="4" hierarchyLevel="2">
  <Value>Austria-Europe</Value>
  <Value>Denmark-Europe</Value>
  <Value>USA-North America</Value>
  <Value>Turkey-Asia</Value>
</DataField>
<DataField name="Location" optype="categorical"
  dataType="string" measure="false"
  numberOfHierarchyLevels="4" hierarchyLevel="1">
  <Value>Vienna-Austria-Europe</Value>
  <Value>Copenhagen-Denmark-Europe</Value>
  <Value>Pittsburgh-USA-North America</Value>
  <Value>Istanbul-Turkey-Asia</Value>
</DataField>
<DataField name="Location" optype="categorical"
  dataType="string" measure="false"
  numberOfHierarchyLevels="4" hierarchyLevel="0">
  <Value>Kettenbrueckengasse-Vienna-Austria-Europe</Value>
  <Value>Winzer Strasse-Copenhagen-Denmark-Europe</Value>
  <Value>Roadstar Avenue-Pittsburgh-USA-North America</Value>
  <Value>Ekin Sokak-Istanbul-Turkey-Asia</Value>
  <Value>Schwarzenbergplatz-Vienna-Austria-Europe</Value>
</DataField>

```

6.5 Query

This part describes the possible OLAP operations, attribute "id" represents the query's identity, attribute "method" indicates the operation method which could be point, sum, min, max and so on.

```

<xs:element name="Query">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Parameter"
        maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required"/>
    <xs:attribute name="method">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="point"/>
          <xs:enumeration value="sum"/>
          <xs:enumeration value="min"/>
          <xs:enumeration value="max"/>
          <xs:enumeration value="avg"/>
          <xs:enumeration value="rollup"/>
          <xs:enumeration value="drilldown"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>

```

The element `<Parameter>` represents the query parameters, the attribute "name" indicates the dimension name, and `<Value>` is the query value for the dimension.

```

<xs:element name="Parameter">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Value" type="xs:string"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

```

The following example represents an aggregation query along the dimension "Location".

```

<Query id="q1" method="sum">
  <Parameter name="Product">
    <Value>TV</Value>
  </Parameter>
  <Parameter name="Location">
    <Value>[ANY]</Value>
  </Parameter>
  <Parameter name="Year">
    <Value>2010</Value>

```

```
    </Parameter>
</Query>
```

6.6 Result

The last part describes the result of OLAP query. Attribute "queryID" indicates the result related query's identity. <Value> contains the resulted value of the query.

```
<xs:element name="Result">
  <xs:complexType>
    <xs:choice>
      <xs:element name="Value" type="xs:string"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:choice>
    <xs:attribute name="queryID"
      type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

Following is an example:

```
<Result queryID="q1">
  <Value>543</Value>
</Result>
```

The full of OMML 2.0 XML Schema Definition can be found in Appendix B.

Chapter 7

Conclusion and Future Work

In this thesis we have presented our development of two multi-tier client systems - OLAP administrator and OLAP query client. Google Web Toolkit (GWT) and Apache Wink client module were applied for development of OLAP query client, which aims at providing an efficient way for the user to send different kinds of queries to the elastic OLAP cloud and receive results. GWT, libraries of Apache Wink, Apache Derby database, OGSA-DAI and Ext GWT were applied for development of OLAP administrator, which inherits some functionality from the OLAP query client and provides means for user with administrator privilege to manage virtual cubes. User with administrator privilege can perform following actions:

- Initiate new virtual cubes
- Delete virtual cubes
- Query the Derby database or OGSA-DAI server to load raw data, transform the raw data into WebRowSet format and submit it to the elastic OLAP cloud

Besides, we have further developed the OLAP Modelling Markup Language (OMML), which was applied as standard communication language between the client systems and the elastic OLAP cloud.

Our research results are also presented in a paper [CY11], which is to be included in proceedings of the International Conference on Cloud and Green Computing (CGC 2011) in Sydney, Australia in December 2011.

For the OLAP administrator, there are following future work possibilities:

1. Till now, we can load raw data either from Derby database or from OGSA-DAI server, but more data sources could be introduced, e.g. support more kinds of relational databases like MySQL and so on, support other data sources like XML database, stream data from a sensor network etc.
2. In the current version, we can transform raw data into WebRowSet XML format and present it in a Ext GWT grid which enables some data preprocess operations like sorting. Actually, possible operations of the Ext GWT grid can be extended,

we can further develop more operations like sum, different kinds of editing. So, the OLAP administrator can better observe the raw data and perform some useful preprocesses before loading the raw data into the OLAP cloud.

3. We could provide more OLAP management possibilities to the OLAP administrator, e.g. graphically manage the hosts which serve for virtual cubes. We could provide drag-and-drop operation possibilities for administrator to graphically create and edit the structure of a virtual cube and its hosts. Besides, In case a virtual cube is deleted, its hosts becomes free, so the administrator could manually put these free hosts to another virtual cube. This operation possibility will save time for initiating new hosts in the cloud and will result in better resource utilization.

For the OLAP query client, there are following future work possibilities:

1. In the current version, the OLAP query client can send a single query at one time to the OLAP cloud. In future, we could enables the client to send series of queries with different parameters and functions at one time, and get series of query results back.

2. As we could have series of query results, we could provide more graphical analysis possibilities to analyze these results and better observe the effect of different query parameters and functions.

Appendix A

WebRowSet XML Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- WebRowSet XML Schema by Jonathan Bruce (Sun Microsystems Inc.) -->
<xs:schema targetNamespace="http://java.sun.com/xml/ns/jdbc" xmlns:wrs="
  "http://java.sun.com/xml/ns/jdbc" xmlns:xs="http://www.w3.org/2001/
  XMLSchema" elementFormDefault="qualified">

  <xs:element name="webRowSet">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="wrs:properties"/>
        <xs:element ref="wrs:metadata"/>
        <xs:element ref="wrs:data"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="columnValue" type="xs:anyType"/>
  <xs:element name="updateValue" type="xs:anyType"/>

  <xs:element name="properties">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="command" type="xs:string"/>
        <xs:element name="concurrency" type="xs:string"/>
        <xs:element name="datasource" type="xs:string"/>
        <xs:element name="escape-processing" type="xs:string"/>
        <xs:element name="fetch-direction" type="xs:string"/>
        <xs:element name="fetch-size" type="xs:string"/>
        <xs:element name="isolation-level" type="xs:string"/>
        <xs:element name="key-columns">
          <xs:complexType>
            <xs:sequence minOccurs="0" maxOccurs="unbounded">
              <xs:element name="column" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="map">
          <xs:complexType>
```



```

        <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="type" type="xs:string"/>
            <xs:element name="class" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="max-field-size" type="xs:string"/>
<xs:element name="max-rows" type="xs:string"/>
<xs:element name="query-timeout" type="xs:string"/>
<xs:element name="read-only" type="xs:string"/>
<xs:element name="rowset-type" type="xs:string"/>
<xs:element name="show-deleted" type="xs:string"/>
<xs:element name="table-name" type="xs:string"/>
<xs:element name="url" type="xs:string"/>
<xs:element name="sync-provider">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="sync-provider-name" type="xs:string"/>
            <xs:element name="sync-provider-vendor" type="xs:string"/>
            >
            <xs:element name="sync-provider-version" type="xs:string"/>
            </>
            <xs:element name="sync-provider-grade" type="xs:string"/>
            <xs:element name="data-source-lock" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="metadata">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="column-count" type="xs:string"/>
            <xs:choice>
                <xs:element name="column-definition" minOccurs="0" maxOccurs="
                    "unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="column-index" type="xs:string"/>
                            <xs:element name="auto-increment" type="xs:string"/>
                            <xs:element name="case-sensitive" type="xs:string"/>
                            <xs:element name="currency" type="xs:string"/>
                            <xs:element name="nullable" type="xs:string"/>
                            <xs:element name="signed" type="xs:string"/>
                            <xs:element name="searchable" type="xs:string"/>
                            <xs:element name="column-display-size" type="xs:string"/>
                        </>
                            <xs:element name="column-label" type="xs:string"/>
                            <xs:element name="column-name" type="xs:string"/>
                            <xs:element name="schema-name" type="xs:string"/>
                            <xs:element name="column-precision" type="xs:string"/>
                            <xs:element name="column-scale" type="xs:string"/>
                            <xs:element name="table-name" type="xs:string"/>
                            <xs:element name="catalog-name" type="xs:string"/>
                            <xs:element name="column-type" type="xs:string"/>

```

```

        <xs:element name="column-type-name" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:choice>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="data">
  <xs:complexType>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element name="currentRow" minOccurs="0" maxOccurs="
unbounded">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="wrs:columnValue"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="insertRow" minOccurs="0" maxOccurs="unbounded
">
        <xs:complexType>
          <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="wrs:columnValue"/>
            <xs:element ref="wrs:updateValue"/>
          </xs:choice>
        </xs:complexType>
      </xs:element>
      <xs:element name="deleteRow" minOccurs="0" maxOccurs="unbounded
">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="wrs:columnValue"/>
            <xs:element ref="wrs:updateValue"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="modifyRow" minOccurs="0" maxOccurs="unbounded
">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="wrs:columnValue"/>
            <xs:element ref="wrs:updateValue"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

Appendix B

OLAP Modelling Markup Language 2.0 XML Schema Definition

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">

  <xs:element name="OMML">
    <xs:complexType>
      <xs:annotation>
        <xs:documentation>OLAP Model Markup Language</xs:documentation>
      </xs:annotation>
      <xs:sequence>
        <xs:element ref="Header" />
        <xs:element ref="ServerInfo" />
        <xs:element ref="DataDictionary" />
        <xs:element ref="Query" maxOccurs="unbounded" />
        <xs:element ref="Result" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="version" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="Header">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Annotation" minOccurs="0" />
        <xs:element ref="Timestamp" minOccurs="0" />
        <xs:element ref="ServiceURI" minOccurs="0" />
      </xs:sequence>
      <xs:attribute name="copyright" type="xs:string" use="required" />
      <xs:attribute name="description" type="xs:string" />
    </xs:complexType>
  </xs:element>

  <xs:element name="Annotation" type="xs:string" />
  <xs:element name="Timestamp" type="xs:string" />
  <xs:element name="ServiceURI" type="xs:string" />
```

```

<xs:element name="ServerInfo">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CubeID" type="xs:string" minOccurs="0"
        maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="DataDictionary">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="DataField" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="numberOfFields" type="xs:integer" use="required"
      />
  </xs:complexType>
</xs:element>
<xs:element name="DataField">
  <xs:complexType >
    <xs:sequence>
      <xs:element name="Value" type="xs:string" minOccurs="0"
        maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required" />
    <xs:attribute name="optype" type="OPTYPE" use="required" />
    <xs:attribute name="dataType" type="DATATYPE" use="required" />
    <xs:attribute name="measure" type="xs:boolean" use="required" />
    <xs:attribute name="numberOfHierarchyLevels" type="xs:integer"
      use="unrequired" />
    <xs:attribute name="hierarchyLevel" type="xs:integer" use="
      unrequired" />
  </xs:complexType>
</xs:element>
<xs:simpleType name="OPTYPE">
  <xs:restriction base="xs:string">
    <xs:enumeration value="categorical" />
    <xs:enumeration value="ordinal" />
    <xs:enumeration value="continuous" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="DATATYPE">
  <xs:restriction base="xs:string">
    <xs:enumeration value="string" />
    <xs:enumeration value="integer" />
    <xs:enumeration value="float" />
    <xs:enumeration value="double" />
    <xs:enumeration value="boolean" />
    <xs:enumeration value="date" />
    <xs:enumeration value="time" />
    <xs:enumeration value="dateTime" />
  </xs:restriction>
</xs:simpleType>

<xs:element name="Query">

```

```

<xs:complexType>
  <xs:sequence>
    <xs:element ref="Parameter" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required" />
  <xs:attribute name="method">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="point" />
        <xs:enumeration value="sum" />
        <xs:enumeration value="min" />
        <xs:enumeration value="max" />
        <xs:enumeration value="avg" />
        <xs:enumeration value="rollup" />
        <xs:enumeration value="drilldown" />
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>
</xs:element>
<xs:element name="Parameter">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Value" type="xs:string" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>
<xs:element name="Result">
  <xs:complexType>
    <xs:choice>
      <xs:element name="Value" type="xs:string" minOccurs="0" maxOccurs="unbounded" />
    </xs:choice>
    <xs:attribute name="queryID" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>
</xs:schema>

```

Appendix C

Class Diagrams of the OLAP Administrator Project



Figure C.1: Class diagram of Administrator, Record1 and Record10

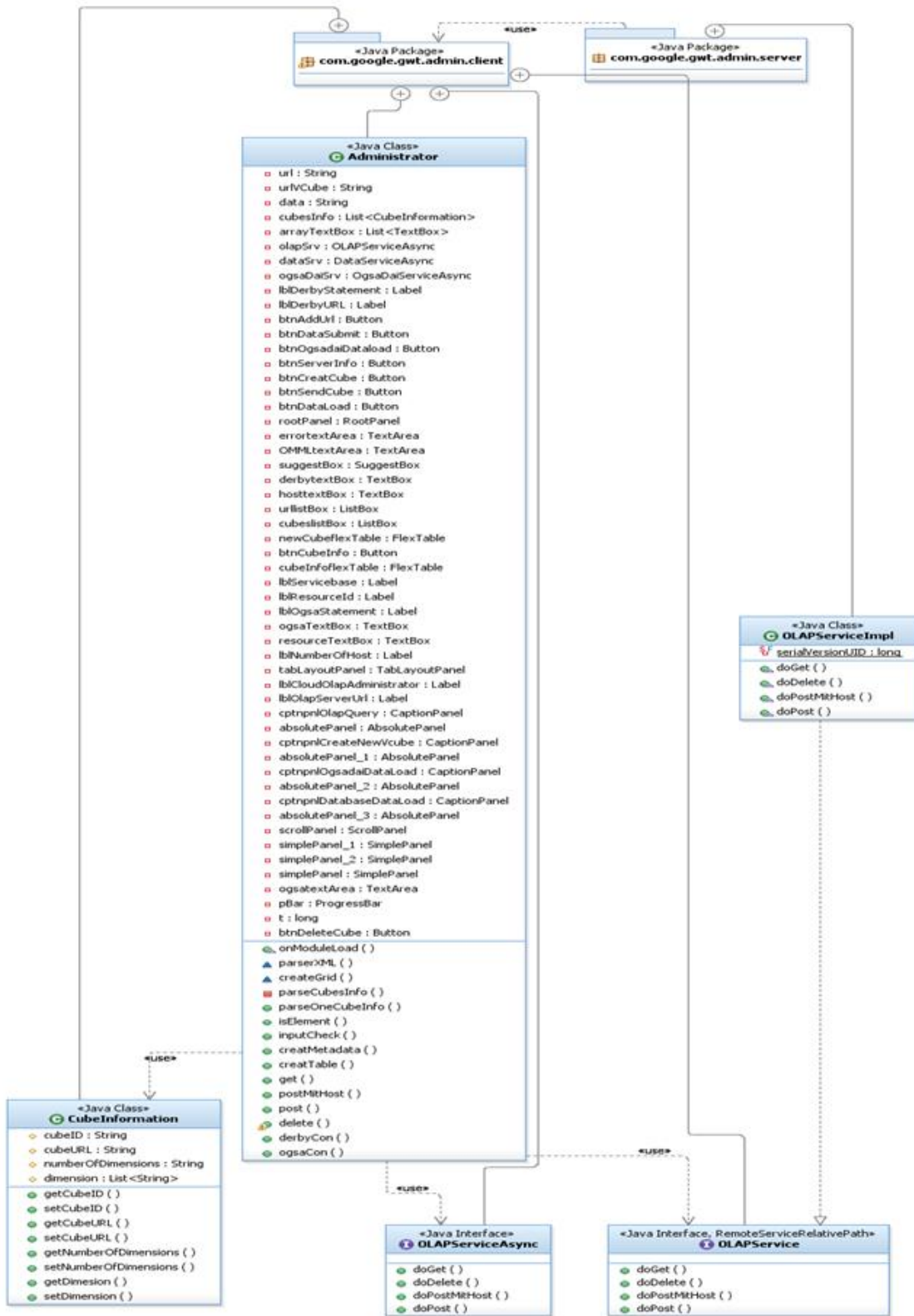


Figure C.2: Class diagram: OLAPService implementation

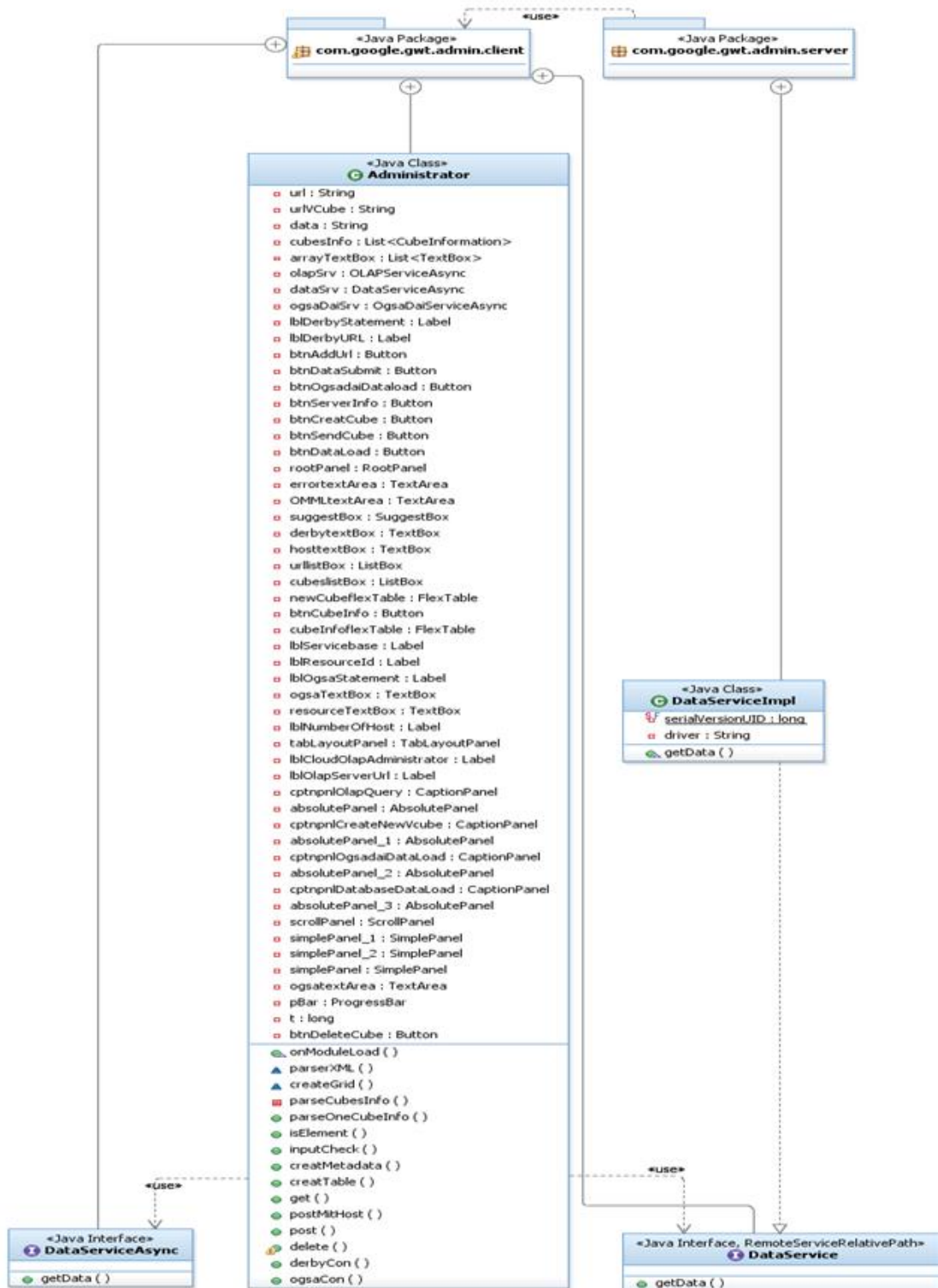


Figure C.3: Class diagram: DataService implementation



Figure C.4: Class diagram: OgsaDaiService implementation

Lebenslauf

Name: Sicen Ye
Geburtsdatum: 29.06.1981
Geburtsort: China

Ausbildung

2003 - 2004 Ein jahr Deutschkurs und Vorstudienlehrgang
2004 - 2009 Bachelorstudium Technische Informatik, Universität Wien
Seit 2010 Masterstudium Scientific Computing, Universität Wien

Bibliography

- [CY11] Peter Brezany Yan Zhang Ivan Janciak Peng Chen and Sicen Ye, *An elastic olap cloud platform*, in proceedings of International Conference on Cloud and Green Computing (CGC 2011) in Sydney, Australia, December 2011. 89
- [Der] Apache Derby, *Derby reference manual*, <http://db.apache.org/derby>. 66
- [EO05] I. Elsayed and U. Onan, *The olap model markup language*, Working Draft, Institute of Scientific Computing, University of Vienna, January 2005. 4, 82
- [FB04a] B. Fiser and P . Brezany, *Olap engine development for distributed parallel computing*, Technical report, institute of Scientific Computing, University of Vienna, February 2004. 12
- [FB04b] B. Fiser and P. Brezany, *Approaches to the development od olap engines*, Technical report, institute of Scientific Computing, University of Vienna, February 2004. 12
- [Fie00] Roy Thomas Fielding, *Architectural styles and the design of network-based software architectures*, Doctoral dissertation, University of California, Irvine, 2000. 1
- [Gar] Jesse James Garrett, *Ajax: A new approach to web applications*, <http://AdaptivePath.com>. 5
- [Goo] Google, *Google web toolkit*, <http://code.google.com/intl/en/webtoolkit>. 5
- [Gri] GridMiner, *Project knowledge grid*, <http://www.gridminer.org>. 1
- [Gro] Data Mining Group, *Predictive model markup language (pmml)*., <http://www.dmg.org>. 4
- [HK00] J. Han and M. Kamber, *Data mining, concepts and techniques*, Morgan Kaufmann, 2000. 76
- [MA05] Malcolm Atkinson el.at. Mario Antonioletti, *The design and implementation of grid database services in ogsa-dai*, Concurrency and Computation: Practice and Experience. Volume 17, April 2005. 32

- [MA09] Armando Fox et al. Michael Armbrust, *Above the clouds: A Berkeley view of cloud computing*, Technical Report No. UCB/EECS-2009-28, 2009. 1
- [oE10] The University of Edinburgh, *Ogsa-dai 4.0 documentation*, <http://ogsa-dai.sourceforge.net/documentation/ogsadai4.0/ogsadai4.0-axis/index.html>, April 2010. 31
- [Ona05] U. Onan, *High performance on-line analytical processing on computational grids*, Master Thesis, 2005. xi, 10
- [Ora] Oracle, *Java enterprise edition: Javasever pages technology*, <http://www.oracle.com/technetwork/java/javase/jsp/index.html>. 18
- [Ora10] Oracle, *Java 2 platform: Standard edition 5.0 api*, <http://download.oracle.com/javase/1,5.0/docs/api>, 2010. 31
- [pro10] Apache Wink project, *Apache wink 1.1 user guide*, <http://incubator.apache.org/wink>, 2010. xi, 25
- [Sen] Sencha, *Ext gwt: Internet application framework for google web toolkit*, <http://www.sencha.com/products/extgwt>. 43
- [Spe06] Keynote Speaker, *Web services at amazon.com*, ICWS '06. International Conference on Web Services, 2006. 1
- [UCI] UCI, *Uc irvine machine learning repository*, <http://archive.ics.uci.edu/ml>. 76
- [W3C04] W3C, *Extensible markup language (xml) 1.0 (third edition)*, <http://www.w3.org/TR/2004/REC-xml-20040204>, February 2004. 2
- [XSt] XStream, *Xstream library to serialize objects to xml and back again*, <http://xstream.codehaus.org>. 22